RADC-TR-90-404, Vol III (of 18)
Final Technical Report
December 1990

THE VERSATILE MAINTENANCE EXPERT SYSTEM (VMES)

Northeast Artificial Intelligence Consortium (NAIC)

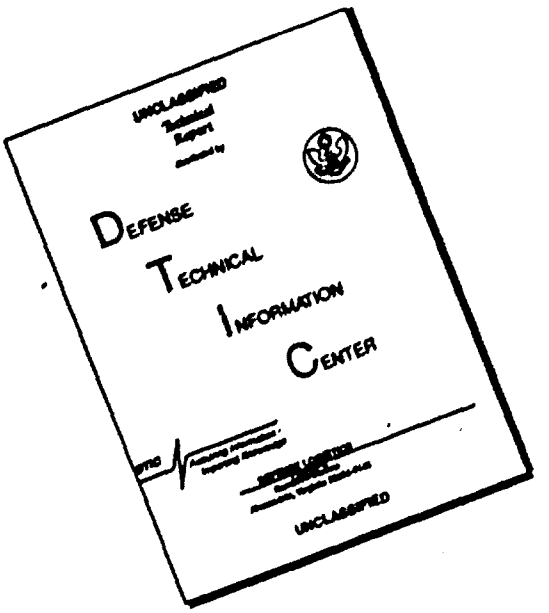
Sargur N. Srihari, Stuart C. Shapiro, Shambhu J. Upadhyaya

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

This effort was funded partially by the Laboratory Director's fund.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-90-404, Volume III (of 13) has been reviewed and is approved for publication.

APPROVED: Dale W. Richards

DALE W. RICHARDS
Project Engineer

APPROVED: John J. Bart

JOHN J. BART
Technical Director
Directorate of Reliability & Compatibility

FOR THE COMMANDER: Igor G. Plonisch

IGOR G. PLONISCH
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (RBES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1990	Final Sep 84 - Dec 89

4. TITLE AND SUBTITLE
THE VERSATILE MAINTENANCE EXPERT SYSTEM (VMES)

5. FUNDING NUMBERS
C - F30602-85-C-0008
PE - 62702F
PR - 5581
TA - 27
WU - 13
(See reverse)

6. AUTHOR(S)
Sargur N. Srihari, Stuart C. Shapiro,
Shambhu J. Upadhyaya

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Northeast Artificial Intelligence Consortium (NAIC)
Science & Technology Center, Rm 2-296
111 College Place, Syracuse University
Syracuse NY 13244-4100

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Air Development Center (COES)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RADC-TR-90-404, Vol III (of 18)

11. SUPPLEMENTARY NOTES (See reverse)
RADC Project Engineer: Dale W. Richards/RBES/(315) 330-3476

This effort was funded partially by the Laboratory Director's fund.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Northeast Artificial Intelligence Consortium (NAIC) was created by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose was to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress during the existence of the NAIC on the technical research tasks undertaken at the member universities. The topics covered in general are: versatile expert system for equipment maintenance, distributed AI for communications system control, automatic photointerpretation, time-oriented problem solving, speech understanding systems, knowledge base maintenance, hardware architectures for very large systems, knowledge-based reasoning and planning, and a knowledge acquisition, assistance, and explanation system.

The specific topic for this volume is the exploration of methods for developing a versatile expert system for equipment maintenance.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Artificial Intelligence, Expert Systems, Fault Detection, Graphical Knowledge, User Interfaces, Hypothetical Reasoning, Semantic Networks	98
	16. PRICE CODE

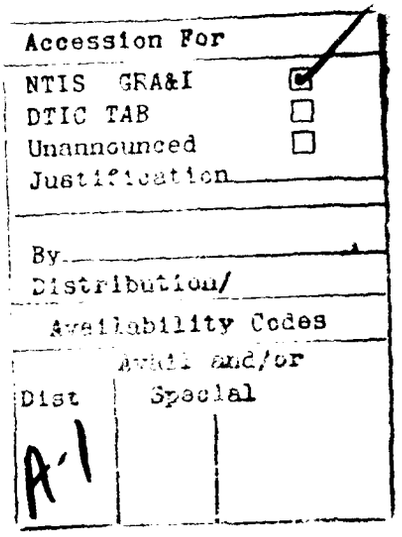
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Block 5 (Cont'd) Funding Numbers

PE - 62702F	PE - 61102F	PE - 61102F	PE - 33126F	PE - 61101F
PR - 5581	PR - 2304	PR - 2304	PR - 2155	PR - LDFP
TA - 27	TA - J5	TA - J5	TA - 02	TA - 27
WU - 23	WU - 01	WU - 15	WU - 10	WU - 01

Block 11 (Cont'd)

This effort was performed as a subcontract by the State University of New York at Buffalo to Syracuse University, Office of Sponsored Programs.



3 THE VERSATILE MAINTENANCE EXPERT SYSTEM (VMES) RESEARCH PROJECT

Report submitted by:

Dr. Sargur N. Srihari, Principal Investigator
Dr. Stuart C. Shapiro, Co-Principal Investigator
Dr. Shambhu J. Upadhyaya, Co-Principal Investigator
Jiah-shing Chen, Graduate Research Assistant
Amruth Kumar, Graduate Research Assistant
Joongmin Choi, Graduate Research Assistant
Sudip Nag, Graduate Research Assistant
Scott S. Campbell, Computer Programmer/Analyst

Department of Computer Science
SUNY at Buffalo
226 Bell Hall
Buffalo, NY 14260

TABLE OF CONTENTS

3 THE VERSATILE MAINTENANCE EXPERT SYSTEM (VMES) RESEARCH PROJECT

3.1 TECHNICAL OVERVIEW

3.1.1 Executive Summary (1984–1989)

The State University of New York at Buffalo started its participation in the Northeast Artificial Intelligence Consortium in 1984 with the objective of developing a versatile expert system for equipment maintenance. A prototype expert system originally projected to be a rule-based system was designed to advise a maintenance technician on testing. However, during the course of this project, it evolved into a more versatile system by incorporating features such as model-based reasoning and communication capabilities such as natural language and graphics. The new system came to be known as the Versatile Maintenance Expert System (VMES).

VMES research is concerned with the development of a system that could diagnose faults in an electronic circuit and interact with a maintenance technician. Versatility has been the main goal of our research. VMES is designed to be versatile across a range of target devices in the circuit domain, across most of the possible faults, across different maintenance levels and across a variety of user interfaces. To achieve these versatilities, the device model-based approach has been followed. VMES has been implemented in SNePS, the Semantic Network Processing System and has several modules: an expandable component library as its knowledge base, an inference package with diagnostic rules, an active database for diagnosis, a user interface for intermediate users to adapt VMES to new devices by incrementally updating the component library, and a multimedia user interface for end users to interact with VMES for fault diagnosis.

Our accomplishments during the lifetime of the project (1984-1989) can be classified into the following seven categories. (1) Device modeling – structural and functional knowledge and efficient representation, (2) Graphical interface for end users to interact with VMES, (3) Model-based reasoning for diagnosis – initial candidate ordering, reordering and elimination, (4) Sequential circuit representation and a general control structure for diagnosis, (5) Representation and diagnosis of a real device, (6) Migration of deep knowledge to shallow knowledge and (7) Enhancements of SNePS, the system used for the implementation of VMES. In the remainder of this summary, a few more details in each are given.

3.1.2 Device Modeling

VMES uses structural and functional descriptions of devices to avoid difficulties of empirical rule-based diagnosis systems in knowledge acquisition, diagnosis capability and system generalization. Based on the requirements of expressibility, buildability, computer-usability and expandability, a device in the circuit domain has been modeled as a hierarchically arranged set of subparts from both logical and physical perspectives. Wires and points of

contact (POCONs) have been explicitly represented in order to perform the diagnosis of faults in circuit connections.

3.1.3 Interfaces

The usability of VMES would be enhanced greatly if it were to communicate with the maintenance technician graphically as well with text. Toward that end, we developed a general theory of "Graphical Deep Knowledge," defined as declarative knowledge that is projectively adequate (adequate for drawing) as well as deductively adequate (adequate for deducing relevant spatial information). We also made major progress in the field of Natural Language Graphics, the attempt to design a knowledge representation that may be used for graphical deep knowledge and for meaning structures that underly the comprehension and generation of natural language. Using these theories, we designed and implemented graphical interfaces for VMES that can draw the devices being tested, and that can graphically show the reasoning process that VMES is pursuing to diagnose the device.

An interface for encoding devices represented using structural templates and instantiation rules has been implemented to facilitate fault diagnosis. This is user-friendly and robust, providing for as few key-strokes from the user as possible. It fills in most of the invariant template, documents the code and stores it in the appropriate file for the user. It was encoded in Franz Lisp on a VAX to begin with, and was transferred to Common Lisp on a TI Explorer.

3.1.4 Model-based Reasoning

A major step in model-based fault diagnosis has been the generation of candidate submodules which might be responsible for the observed symptom of malfunction. After the candidates are determined, each submodule can then be examined in turn. It is useful to be able to choose the most likely candidate to focus on first so that the faulty parts can be located sooner. We have developed a systematic method for candidate ordering that takes into account the structure of the device and the discrepancy in outputs between the observed and expected values. However, because the same good/bad output pattern of a device always gives rise to the same initial ordering, the method has its limitation. For any device and good/bad output pattern, it is easy to come up with an example on which the method does poorly in the sense that the actual faulty part is in the last place of the initial ordering.

To fix this problem, more dynamic methods, candidate reordering and elimination, have been developed. Both methods modify the candidate list as new information becomes available. Reordering moves components connected to the inconsistent inputs of the current candidate to the front of the candidate list and those connected to the consistent inputs to the tail. Elimination removes components which no longer have any non-error-propagating paths to incorrect primary outputs after the current candidate is found not error-propagating. It has been proved that under the single fault assumption, the average

number of components to checked is $O(\log n)$, where n is the number of components in the circuit under test. And in general, $k \log n$ components will be checked when there are k faults in the circuit. As to the implementation, all the above-mentioned theories have been implemented in our VMES system. Also the system has been successfully ported from our UNIX machine using SNePS-79 and Franz Lisp to TI Explorers using SNePS-2 and Common Lisp. The improvement in preformance is enormous. It runs at least ten times faster than before.

3.1.5 Sequential Circuit Diagnosis

In order to incorporate sequential circuit diagnosis into VMES, the following steps have been taken: (1) change in device knowledge representation; (2) change in control structure and inclusion of assumption relaxation; and (3) candidate generation based on electrical behavior. The change in representation was essential for better organization of the device knowledge and the incorporation of sequential components. Devices no more have any physical hierarchy, and logical hierarchy is not related to physical details of the device. This has enabled arbitrary number of logical levels, and has allowed arbitrary grain of focus for diagnosis. Wires were no more represented explicitly. Diagnosis of wires and POCONs was intended to be hard-coded into VMES. Assignable variables were found to be necessary to represent states in sequential components. Since SNePS is built on the philosophy of logical programming, it did not allow for such variables. Hence, the new scheme of representation had been developed in CommonLisp.

The control structure has been streamlined so as to accept various options such as shallow reasoning and diagnosis with multiple symptoms. Explanation is a very necessary part of an expert system and hence, an explanation generation system was also incorporated. The criteria upon which the system should discard single-fault and non-canceling fault assumptions has been worked out and partially implemented. The new system VMES II, with the above modifications, runs considerably faster than the previous system.

Since sequential circuit diagnosis is harder than combinational, the stage of candidate generation had to be exploited to the maximum so as to narrow the focus of diagnosis as much as possible. Therefore, candidate generation based on electrical behavior was researched and implemented. In this method, given the symptom, the system works backward through the subdevices to come up with a probable candidate list. The scheme is based on the assumption that one could shorten the suspect-list by eliminating those of the relevant inputs that can in no way contribute to the observed symptom at output. This knowledge can be conveniently expressed for small devices as fault characteristics and was exploited during diagnosis of large systems.

Sequential circuit diagnosis poses many special problems. A sequential circuit has feedback loops, and, therefore, it is necessary to specify the start and end of the loops for proper simulation of states. Moreover, during simulation and candidate generation, more than one (as many as the clock cycles) value has to be stored with each device-port. Fault characteristics are not as well behaved for sequential circuits and are to be specified for the

superdevice too. These and many other related problems have/are being solved towards incorporating sequential circuits.

3.1.6 Diagnosis of Printer Buffer Board

Although VMES was intended to be adaptable to a wide range of devices in the domain of digital circuits, selection of a realistic device was essential as a test bed. Upon recommendation by RADC, a Heathkit Printer Buffer Board was selected. This test device, assembled locally, is of reasonable complexity to work with. It consists of an eight-bit microprocessor, two sets of serial and parallel ports, memory and latches. This device has been represented hierarchically at various levels of abstraction to facilitate diagnosis. A device of this kind helped spur new ideas, extensions and refinements to the diagnosis theory.

3.1.7 Migration of Deep Knowledge to Shallow Knowledge

Deductive reasoning systems, including automatic fault diagnosis, usually make use of vast numbers of rules to infer new knowledge from existing knowledge. Different rules may be at different levels of generality. In particular, a method of rule nesting enables a rule to contain a subrule in its consequent. If some instance of the antecedent of the outer rule is satisfied, that instance of the subrule will be asserted in the knowledge base. This subrule has a lower level of generality than its dominating rule. We call this phenomenon the migration of deep to shallow knowledge.

As more specific knowledge emerges from the general knowledge, the speed of inference might become slower if the system tries to use both specific and general rules for similar problems. We need a proper scheme to take advantage of specific knowledge to avoid duplicate inference branches and, in the long run, to realize fast inference. An idea of shadowing general knowledge has been suggested and implemented by exploiting its instance information accumulated in the previous reasoning. A list of instances containing the binding information was maintained for each rule so that this rule can be blocked in the next inference if instances of that rule are already activated with the same binding information. This shadowing strategy is tested for wire-faulty detection rules of the M3A2 circuit, and the result shows a significant improvement in diagnosis speed for the same types of wires.

3.1.8 Enhancements of SNePS

During the course of this project, we have continued the development of the Semantic Network Processing System. This effort resulted in a new major version of the system called SNePS-2.0, and a new minor version, called SNePS-2.1. SNePS-2 represents a major redesign of SNePS, and is implemented in Common Lisp, for maximal portability. SNePS-2.1 incorporates an assumption-based belief revision system (SNeBR) as a standard feature of the system. SNeBR is designed to reason the consistency of rules and hypotheses defined within a particular context or belief space. SNeBR was applied to fault detection

in electronic circuits [Campbell and Shapiro, 1986]. We have ported SNePS to, and have maintained it on over seven different computer systems.

3.1.9 Possible Extensions

The current VMES system can be enhanced in a number of ways. The control structure can be generalized to include various schemes of diagnosis. For instance, a retry method, or direct isolation or intersection isolation can be applied in a sequence to diagnose a circuit. This involves the development of direct and/or intersection isolation techniques and its implementation. Model-based reasoning can be enhanced by incorporating heuristic knowledge in the reasoning process. It is also worthwhile to investigate the effect of multiple tests, test generation techniques, and mixing of procedural and declarative knowledge in diagnosis. The versatility of VMES can be further enhanced by the development and integration of analog circuit component diagnosis. Diagnosis of intermittent faults is another area that needs further investigation.

3.2 DEVICE REPRESENTATION

3.2.1 Introduction

A versatile fault diagnosis system is an expert system that is capable of diagnosing a variety of faults on a wide range of devices in its domain. In this work, issues about knowledge representation for versatile fault diagnosis are investigated with digital circuits as the experimental domain. This domain consists of numerous types of devices with different configurations and functionalities. Automatic diagnosis of faults in digital circuits is highly desirable due to their widespread use and relative short market life cycles as well as the general shortage of qualified maintenance technicians. Our aim is to develop a knowledge representation formalism which is capable of supporting a versatile fault diagnosis system while still retaining the effectiveness of fault diagnosis.

Versatility in diagnosis is multifold: ability to adapt to different devices without extensive knowledge engineering; capability of diagnosing a wide range of common faults; capability of operating at different maintenance levels; and capability of interacting with the user through various media.

Device representation is the task of modeling a device and abstracting it into a representation formalism suitable for a specification-based system. As knowledge engineering is to empirical-rule-based systems, device modeling/representation is the key to the success of a device-model-based fault diagnosis system, since its major power comes from knowledge about the structure and function of a device. While much work has been done in the area of knowledge engineering for empirical-rule-based systems [Feigenbaum, 1979, Quinlan, 1982, Waterman, 1979, Williams, 1986], little has been done in device representation for device-model-based systems, and most current device representation schemes are either improper or insufficient for fault diagnosis. Computer hardware description languages such as ISP [Siewiorek, 1974a], PMS [Siewiorek, 1974b] and Zeus [German and Lieberherr, 1985] are designed for communication between computer designers, but not for the purpose of fault diagnosis [Barbacci and Uehara, 1985, Chu, 1974]. Other existing device description schemes for fault diagnosis are either *ad hoc* or insufficient to support a versatile fault diagnosis system. Examples are the predicate logic representation used by Genesereth [Genesereth, 1984] and the schematic diagram representation language designed by Davis [Davis and Shrobe, 1983].

3.2.2 Knowledge of Versatile Diagnosis

It has been argued that the performance of an expert system depends mostly on the contents and the forms of its knowledge, since it is the major resource of its reasoning [Buchanan and Shortliffe, 1984, Hayes-Roth *et al.*, 1983, Michie, 1980]. In this section, we first analyze the knowledge for troubleshooting electronic circuits, and identify the core knowledge for versatile fault diagnosis. We discuss how domain experts coordinate different views of a same

device in diagnosing and repairing it, and the difficulties they might have in coordinating the different views.

Knowledge Analysis and Characterization

In analyzing the knowledge used in circuit diagnosis, we suggest that the knowledge be categorized as device-specific empirical associations, generic domain knowledge, and a device model.

Device-specific empirical associations relate observed symptoms to possible fault hypotheses for a specific device. Such rules are highly device specific, and cannot apply to other devices in the same domain, since devices in the same domain may have different structures.

Generic domain knowledge is the general knowledge in the designated domain that domain experts use for fault diagnosis. A very primitive piece of generic knowledge for circuit fault diagnosis is that "when an output of a device deviates from the expected value regarding to the known inputs, it implies that the device is malfunctioning". Another one is that "to locate the faulty component(s), one should first identify the signal flow paths from inputs to the bad output, this can be efficiently done by back-tracing the connections from the bad output to the relevant inputs". The domain knowledge is not necessarily very primitive, it may also relate to higher human perception as in the example of "a burnt appearance of a component implies that the component is a potential faulty part".

A model of the target device, which consists of the structural and functional information about the device, is maintained by the technician when troubleshooting electronic circuits. Though the model is device specific, unlike device-specific empirical associations, a model of a device is "public" knowledge which is highly structured and readily available at the time when the device is designed. This model is sometimes referred to as a "mental model" of the device since it is the technician's view of a device when troubleshooting it [Rasmussen and Jensen, 1974]. This model is also referred to as a "design model" of the device, since it usually reflects, though not always necessarily, the design of the device [Genesereth, 1982].

Core Knowledge of Versatile Fault Diagnosis

In analyzing the knowledge used by maintenance technicians for troubleshooting electronic equipment, it turns out that human experts use all kinds of knowledge described above, viz., device-specific empirical associations, generic domain knowledge, and a model of the target device, in an intermixed manner.

In attempting to combine all knowledge to facilitate diagnosis, a two-level architecture has been proposed for neurological diagnosis [Xiang and Srihari, 1986], which suggests that the system that the system first works on the empirical-association-based module, and then turns to the device-model-based module if the problem can not be successfully solved by the first module. Two reasons convince us that we will not adopt this idea. First, human experts usually use all sorts of their knowledge in an interleaving or mixed manner for fault diagnosis. A sharp partition of knowledge into two separate working modules seems

unrealistic to us. Second, our research is to develop a device representation scheme to support a versatile fault diagnosis system, and the inclusion of empirical associations at this point may impair the construction of such a versatile system.

As mentioned before, versatility of an automatic fault diagnosis system is extremely important in an electronic circuit domain due to the fast rate at which new products are introduced and their relatively short market lives. In noticing that an experienced technician is able to effectively troubleshoot an electronic device by using the schematic diagrams of the device and his general knowledge about troubleshooting devices in the domain and without having to learn how the device may fail, we define an automatic versatile fault diagnosis system as an expert system which behaves like an experienced technician who is competent in diagnosing devices he has never seen before. A major point here is that a versatile fault diagnosis system should be able to be adapted to new devices easily, just like an experienced technician does.

Device-specific empirical associations are quite different from the knowledge of a device model in both the contents and the representation forms. Device-specific empirical associations are assertive knowledge (or propositions) relating symptoms to possible faults. It is natural to represent them as production rules. Knowledge of a device model is basically descriptive, and can be best represented as semantic networks or as frames. There are two major hurdles in including the device-specific empirical associations in a versatile fault diagnosis system: techniques in acquiring it by interviewing with domain experts through knowledge engineering have to be improved so that this process will not slow down the adaptation of the system to other devices; and the capability of an expert system in selecting and using proper knowledge at proper time from a knowledge-base (or knowledge-bases) containing various types of knowledge in different forms has to be achieved so that the system can have an acceptable performance.

One major consideration in developing a versatile fault diagnosis system is the system's ability in adapting itself to other devices. It is improper to include the device-specific empirical associations in a versatile maintenance system, since this may impair system versatility, and moreover, this kind of knowledge is not available at all for newly designed devices. Therefore, only the generic domain knowledge and the knowledge of a device model should be incorporated into a versatile fault diagnosis system. In mimicking the versatility of experienced maintenance technicians in troubleshooting devices they had never seen before, the generic domain knowledge is transformed into the search algorithms and diagnosis rules of the fault diagnosis system, and the knowledge of a device model, which is the basis of the system's reasoning, becomes the core knowledge of the system.

3.2.3 Logical and Physical Knowledge in Diagnosis

The emphasis of most previous research on the device-model-based approach to fault diagnosis has been on using the logical structure of a target device. Such a representation emphasizes the functional interrelations of components but not the physical interrelationships, e.g., functionally unrelated components may be physically related (adjacent, in the

same area, etc.). However, knowledge about physical device structure often plays an important role in fault diagnosis performed by human technicians. This research explores the representation and use of knowledge about both logical and physical structures of target devices in a versatile maintenance system. In particular, we examine the relationships (cross-links) between logical and physical structures.

The use of physical structure in a diagnostic problem in the medical domain, viz., neurological diagnosis based on a model of neural pathways in the human spinal cord, was explored by Xiang and Srihari [Xiang and Srihari, 1985]. In their system, two functionally unrelated paths may be examined due to their physical proximity. In the domain of circuit diagnosis there is little in the literature on physical structure representation with the exception of references made by Davis [Davis and Shrobe, 1983, Davis, 1984]. He suggests including a physical description based on the notion that different paths of interaction or adjacency should be represented to diagnose different kinds of faults. A particular application of utilizing the physical structure description of the device is demonstrated as the diagnosis of bridge faults under the assumption that bridges can only occur between two adjacent pins of an IC (integrated circuit) chip. Although the result is effective and impressive, there are two limitations. First, human experts can visually locate a bridge fault easily by exhaustively searching (looking around) the suspected local area without the complicated reasoning i.e. suggested. When computer vision techniques are unavailable, it may be proper to rely on the user to pinpoint the bridge fault without the strict assumption that bridge faults can only occur between two adjacent pins of an IC chip. Based on this argument, a better role of the automatic fault diagnosis system is to locate and direct the user to the suspected area. Moreover, we find that human experts do maintain a model of the physical structure of the device being diagnosed, but they use it in a more general manner, which is not limited to troubleshooting a bridge fault.

Human diagnosticians for electronic devices seem to simultaneously maintain models of the logical and physical structures of the target device. They carry out most of the diagnostic reasoning over the logical structure of the device due to its functional association. While carrying out the reasoning, the logical structure is apparently mapped to the physical structure from time to time. Tests and measurements are first initialized using the logical structure, and then are realized and executed on the physical structure. Repair, which is usually done by replacing a physical unit or by fixing a physical connection, is planned and done on the physical structure. In other words, maintenance technicians use a model of physical structure of the target device, which is a hierarchically arranged set of replaceable physical components at various maintenance levels such as field-level and depot-level. By mapping the logical structure of the device to its physical equivalent, maintenance technicians are able to terminate the diagnostic process at the right moment and to form an adequate repair plan.

Given that the mapping between the logical structure of the device and its physical equivalent happens throughout the diagnostic process at all hierarchical levels, the speed in carrying out the mapping is critical to the time needed to locate faults. This implies that objects on both the logical level structure and the physical structure of the device should

be closely linked to each other so that the mapping is done efficiently. Even experienced technicians may have difficulty in locating a point of a schematic diagram on the real device, where the schematic diagram represents the logical structure of the device, and the form of the real device is the physical structure; which is attributable to a lack of cross-links at all hierarchical levels of the device in human memory. On the other hand, when modeling and representing a device in an automatic fault diagnosis system, the cross-links between its logical structure and physical structure can be modeled and represented to an appropriate level of detail.

3.2.4 Structural Representation

In our system, a device is abstracted as a hierarchically arranged set of objects, and each object is abstracted at two levels. At level-1 abstraction, an object is modeled as a module with ports; and at level-2 abstraction, the structures of the object is envisioned. An object is represented according to these two abstraction levels from both logical and physical perspectives. Abstractions of an object at these two levels are represented by SNePS rules and SNePS assertions. The former are categorized as *instantiation rules* and the latter as *structural template*. The representation for cross-links between the logical structure and the physical structure is also discussed.

Instantiation Rules as the Level-1 Abstraction

At level-1 abstraction, knowledge about a component type is represented as a SNePS rule. The rule is used later on to instantiate an object of the component type as a module with its own ports and associated functional descriptor. The functional descriptor contains information about the functional description of the component type. The instantiation rule for a physical component type is a little bit simpler in that it contains no functional information of the component type.

Logical Structure

The instantiation rule for objects of the M3A2 type is shown in the SNePSUL (SNePS User Language) command form in Figure 3.2.1. The first three lines of the instantiation rule says that "if x is an M3A2-type object, which is a logical object, and it is to be instantiated at its level-1 abstraction (IRfL1), then do the following". The next five "cq's" will instantiate the ports of the object when this rule is executed. I/O ports of an object are the places where the input/output values of the object are stored. Measured (observed) I/O values depict the real behavior of the object, and calculated I/O values show its expected (normal) behavior. The last two "cq's" create the functional descriptors of the object; functional descriptors are pointers to the representation of the function of the object. The first one says "in order to simulate (calculate) the value of the first output, use the function M3A2out1 which takes three parameters, viz., the inputs of the object x in order". The "tolrnc" denotes the tolerance allowed for a measures value when compared to the calculated value. This is

```
(build avb $x
       ant (build object *x type M3A2 abs-lv IRfL1 modality logical)
       cq (build modality logical
                 object (build type I-PORT port-of *x id inp1
                                signal (build type D bit-width 2))) = vINP1
       cq (...)
       cq (...)
       cq (...)
       cq (build modality logical
                 object (build type O-PORT port-of *x id out2
                                signal (build type D bit-width 5))) = vOUT2
       cq (build object *vOUT1 sfunc M3A2out1
                 tolrnc 0 pn 3 p1 *vINP1 p2 *vINP2 p3 *vINP3)
       cq (...))
```

Figure 3.2.1: Instantiation rule for M3A2 type objects

especially important for analog components, and is usually set to zero for digital devices. Similar functional descriptors can be included for the input ports if the inference of input value from outputs is desired (these are not shown in the figure).

Physical Structure

The instantiation rule for objects of the MAC3200 type is shown in Figure 3.2.2 "MAC3200" is the physical equivalent of the logical component type "M3A2". The first three lines of the instantiation rule says the "if x is an MAC3200-type object, which is a physical object, and it is to be instantiates at its level-1 abstraction (IRfL1), then do the following". The next twenty "cq's", which are shown in partial to save space, will instantiate the

```
(build avb $x
       ant (build object *x type MAC3200 abs-lv IRfL1 modality physical)
       cq (build modality physical
                 object (build type P-PORT port-of *x id 1))
       cq (build modality physical
                 object (build type P-PORT port-of *x id 2))
       .....
       cq (build modality physical
                 object (build type P-PORT port-of *x id 20)))
```

Figure 3.2.2: Instantiation rule for MAC3200 type objects

twenty ports of the object when this rule is executed. The instantiation rule for a physical component type is quite similar to that for a logical type component type except that all ports of a physical object are P-PORTs, which are functional (logically) neutral, and thus no functional descriptors are associated with these ports.

Structural Templates as the Level-2 Abstraction

At level-2 abstraction, a structural template, which is implemented as a SNePS assertion, is used to describe the subparts of a logical object at the next hierarchical level, and the wire connections between the object and its subparts, as well as those among the subparts themselves. Since wires are eliminated from the physical abstraction, the structural templates of a physical component type only contain descriptions of its non-wire subparts.

Logical Structures

The structural template representation is shown in Figure 3.2.3. The representation can be viewed as consisting of five parts—an identification section, a subparts section, a connections section, a part-links section. The last two sections in a structural template, whose contents are missing in the above SNePSUL command, concern the cross-links between the logical structure and the physical structure of a device, and are discussed later.

The identification part, which consists of the first three lines of the SNePSUL "build" command if Figure 3.2.3, denotes that the representation is the structural template for the logical component type M3A2 at the level-2 abstraction (STfL2). The subparts section describes the subparts of the component type at the next hierarchical level. A new case-frame, "id/type", is introduced to describe the subparts of a logical component type within its structural template. The "id" is composed of the name of the component type, i.e., M3A2, and a unique id, such as M3, A1, and W6, within the component type. It identifies the subpart in the rest of the structural template; it also serves for name extension of the subpart when it gets instantiated. For instance, if D1 is an M3A2 device, and its first subpart, which is identified as M3A2-M1 in the structural template, is being instantiated, the subpart will be instantiated with a name of "D1-M1". The part "type" of the subpart specifies its component type; this information is needed when the subpart is to be instantiated. The connections section of the structural template specifies the connections. Note that connections by port superimposition and by POCON (point of contact) are treated differently as discussed in [Taie and Srihari, 1987].

A structural template provides the necessary knowledge about the sub-structure of all objects of same component type without representation overhead. Unlike instantiation rules, structural templates are never executed (fired) to produce a representation for any specific object. When reasoning on the sub-structure of an object is required, instead of instantiating the sub-structure (all the subparts and connections) and then reasoning on the resulted representation, we do it directly on the structural template of the object. If suspicious subparts are located, they (but not all subparts) instantiated at their level-1 abstractions using proper instantiation rules for further examination.

```
(build

  type M3A2
  abs-lv STfL2
  modality logical

  sub-parts ((build id M3A2-M1 type MULT)
             (build id M3A2-M2 type MULT)
             (build id M3A2-M3 type MULT)
             (build id M3A2-A1 type ADDER)
             (build id M3A2-A2 type ADDER)
             (build id M3A2-W1 type WIRE3)
             (build id M3A2-W2 type WIRE3)
             (build id M3A2-W3 type WIRE3)
             (build id M3A2-W4 type WIRE2)
             (build id M3A2-W5 type WIRE3)
             (build id M3A2-W6 type WIRE2)
             (build id M3A2-W7 type WIRE2)
             (build id M3A2-W8 type WIRE2))

  connections
  ((build equiv (findorbuild type B-PORT port-of M3A2-W1 id 1
                             signal (findorbuild type D bit-width 2))
          equiv (findorbuild type I-PORT port-of M3A2 id inp1
                             signal (findorbuild type D bit-width 2)))
   .....
   (build contact (findorbuild type B-PORT port-of M3A2-W2 id 2
                               signal (findorbuild type D bit-width 2))
          contact (findorbuild type I-PORT port-of M3A2-M1 id inp1
                               signal (findorbuild type D bit-width 2)))
   .....)

  part-links (.....)

  port-links (.....))
```

Figure 3.2.3: Structural template for M3A2 type objects

```
(build

  type MAC3200
  abs-lv STfL2
  modality physical

  sub-parts ((build id MAC3200-U1 type MC00 mntn-lv DEPOT)
             (build id MAC3200-U2 type AC00 mntn-lv DEPOT)
             (build id MAC3200-U3 type MC00 mntn-lv DEPOT)
             (build id MAC3200-U4 type AC00 mntn-lv DEPOT)))
```

Figure 3.2.4: Structural template for MAC3200 type objects

Physical Structure

The structural template representation is shown in Figure 3.2.4. Unlike the structural template for a logical type, which consists of five sections, the structural template for a physical component type has only two component sections: the initial section and the subparts section. This is because the wires are eliminated from the physical representation of a device, thus no connection is to be specified, and because the cross-links between the logical structure and the physical structure have been specified at the structural template of the logical component type or elsewhere as will be discussed later.

The identification part, which is the first three lines of the SNePSUL "build command", denotes that the representation is the structural template for the physical component type MAC3200 at its level-2 abstraction (STfL2). The subparts section describes the subparts of the component type at the next hierarchical level. A new semantic network case-frame, "id/type/mntn-lv", is introduced to describe the subparts of a physical component type within its structural template. The meaning and use of part "id" and part "type" are the same as those in a structural template for a logical component type as described in the last section. The "mntn-lv" indicator shows the intended maintenance level of the subpart, i.e., the maintenance level where the subpart, if found faulty, is replaced without further diagnosis. These informations are used for instantiating a physical subpart.

Knowledge about the intended maintenance level is associated with the physical structure of a device because the repairment of a device is performed based on the physical model of the device. It is adequate to store the "mntn-lv" (maintenance level) tag of an object at the subpart section of the structural template of the object's super-part. A more straightforward way is to store the "mntn-lv" tag at the instantiation rule of each component type, but this may cause problems. The reason is that "'mntn-lv" values of objects with same component type may be different when they are used in different devices. For instance, as AC00 chip (an adder) may have a different "mntn-lv" value of DEPOT when it is a subpart of Air-Force-Device-1, and have a different value as FIELD when it is a subpart of

Navy-Device-3. This implies that the "mntn-lv" value of an object is not only complexity dependent but also environment-sensitive, and thus it should be stored at the subpart section of structural templates rather than at the instantiation rule. Though currently, only FIELD and DEPOT levels are used in VMES, "mntn-lv's" and the corresponding system parameter VMES-IML, which stores the intended maintenance level of a diagnostic session, can be set to any arbitrary maintenance level by the user if desired.

Cross-Links between Logical and Physical Structures

There are two kinds of cross-links between the logical and physical structure of a device. The first kind is the cross-links for components. The second kind is the cross-links for ports. Like representing the level-2 abstraction of a device for its sub-structures, the cross-links between the logical and the physical structures is implemented as structural templates to remove any representation redundancies. The cross-links for components are specified in the part-links section of the structural template of the logical object, and the cross-links for ports are specified in the port-links section as partially shown in Figure 3.2.5.

3.2.5 Functional Representation

The function of an object in the electronic domain can be best abstracted as the relationship between its inputs and outputs. The functional description should be usable to simulate the component behavior, i.e. to calculate the values of output ports if the values of the input ports are given. It should also be usable to infer the the values of the input ports in terms of the values of other I/O ports. This is important if hypothetical reasoning is used for fault diagnosis. Though at this stage, VMES only uses the functional description to calculate values at output ports, our representation scheme can be used both ways.

As depicted by the instantiation rule for M3A2 type, a functional descriptor of a port contains a pointer to its functional description as well as other information concerning the use of the functional description. The functional description itself is implemented as a LISP function which calculates the desired port value in terms of the values of other ports. Every port of a component type has such a function associated with it. The functional descriptions for the output ports of the component type M3A2 are shown in Figure 3.2.6.

Some different ports of different component types might have the same function, some functions can be shared. For instance, the simple function "ECHOback", which simply returns its input, can be shared by several different component types, viz., by the type "super-buffer", the type "wire" and the type "one-to-one transformer". All these component types show the same behavior at out level of component abstraction: they echo the input to the output.

```
(build

  type M3A2
  abs-lv STfL2
  modality logical

  sub-parts (.....)

  connections (.....)

  part-links ((build object M3A2-M1 inside MAC3200-U3)
              (build object M3A2-M2 inside MAC3200-U3)
              (build object M3A2-M3 inside MAC3200-U1)
              (build object M3A2-A1 inside MAC3200-U4)
              (build object M3A2-A2 inside MAC3200-U2))

  port-links ((build equiv (findorbuild type I-PORT port-of M3A2-M1 id inp1
                                        signal (findorbuild type D
                                                            bit-width 2))
                       equiv (findorbuild
                              bit (findorbuild type P-PORT
                                               port-of MAC3200-U3
                                               id 1)
                              lo-bit (findorbuild
                                      bit (findorbuild
                                           type P-PORT
                                           port-of MAC3200-U3
                                           id 3))))
  .....))
```

Figure 3.2.5: Cross-links between logical and physical structures

```
(defun M3A2out1 (inp1 inp2 inp3)
  (plus (product inp1 inp2)
        (product inp1 inp3)))

(defun M3A2out2 (inp1 inp2 inp3)
  (plus (product inp1 inp3)
        (product inp2 inp3)))
```

Figure 3.2.6: Output functions for M3A2 type objects

3.3 INTERFACES

3.3.1 Natural Language Graphics[1]

Introduction

Natural Language Graphics (NLG) deals with diagram generation driven by natural language utterances. This investigation applies the methods of declarative knowledge representation to NLG systems. Declarative knowledge that can be used for display purposes as well as reasoning purposes is termed "Graphical Deep Knowledge" and described by supplying syntax and semantics of its constructs. A task domain analysis of Graphical Deep Knowledge is presented covering forms, position, attributes, part, parts, classes, reference frames, inheritability, etc.

Part hierarchies are demonstrated, and criticism of traditional inheritance-based knowledge representation formalisms is derived from this finding. The "Linearity Principle of Knowledge Representation" is introduced and used to constrain some of the presented knowledge structures. The analysis leading to Graphical Deep Knowledge also results in the description of two fundamental conjectures about knowledge representation.

The Gricean maxims of cooperative communication are used as another source of constraints for NLG systems. A new maxim for technical languages is introduced, the "Maxim of Unnecessary Variation". It is argued that common symbolic representations like circuit board diagrams have not yet been described in literature by explicit feature analysis, and that this in necessary to give a system knowledge about the meaning of the diagram it is displaying.

Part of the developed theory has been implemented as a generator program that creates pictures from knowledge structures and as an ATN grammar that creates knowledge structures from limited natural language input. The function of the picture generation program (TINA) as a user interface for a VMES has been demonstrated. An older version of TINA incorporates a module of "Intelligent Machine Drafting" (IMD), a completely new subfield of AI that has been introduced in this research and that relates to Computer Aided Design (CAD). The IMD program does layout and routing for the members of a simple class of functional circuit diagrams based on a policy of symmetry and equal distribution over the available space. This layout/routing is based on cognitive requirements as opposed to physical requirements used by CAD systems.

[1] This section is a condensation of a paper by J. Geller, "Natural Language Graphics for Human Computer Interaction" Submitted to the *International Journal of Man-Machine Studies*.

Representational Constructs of Graphical Deep Knowledge

Form Knowledge

A number of different scientific subfields and fields have been interested in the representation of forms. Among these are computer vision, computer graphics, and imagery, but also solid modeling computer aided design (CAD) and character recognition. We argue [Geller, 1988] that no representation in any of these fields satisfies the requirements for graphical deep knowledge. These requirements are:

- The representation should be projectively adequate.
- The representation should be deductively adequate.
- The representation should be based on conceptual primitives that seem natural to the human observer.
- The representation should support relations between primitives which are natural to humans.
- The representation may contain redundant information.

To fulfill these requirements a representation that consists of basic forms (icons) and asserted relations is used. The basic forms are (supposed to be) meaningful to human observers. Every basic form is represented as a procedure that has three properties. *(1)* The procedure consists of calls to graphics primitives. *(2)* Executing a procedure of the name <name> results in the drawing of an object that is described by <name>. *(3)* The procedure <name> is accessible as a concept in the knowledge representation system, *i.e.*, it functions simultaneously as a node in a semantic network. The representation of a basic form is therefore projectively adequate and also a conceptual unit. Relations between icons are represented propositionally.

The SNe^{D}S system is used in the following way to accommodate the described form representation. The name of every basic form in the system is a base node in the SNePS semantic network. The SNePS inference machine treats it as a conceptual unit and permits reasoning about it. At the same time every SNePS node is also an uninterned LISP atom. This atom refers[2] to a LISP function made up of calls to graphics primitives from a LISP graphics package. Objects and forms are separate nodes, linked by an asserted proposition. This conceptual separation of forms and objects makes it possible to associate a form with a class of objects, instead of a single object.

Part Hierarchies

Part hierarchies have been of fundamental importance in a number of different areas of artificial intelligence. Knowledge representation has dealt with them as well as hardware modeling in maintenance and research in computer vision.

[2]The linkage of the function has been handled differently depending on the dialect of LISP used. Our favorite solution has been to use the function cell of an interned atom of the same name as the node.

Our interest in part hierarchies is motivated by the need for a method to decide *"what content"* to put on the screen of an NLG system and *"how to organize it,"* to be optimally useful to a viewer. In KBUIMS (knowledge based user interface management system) design this complex of problems has been referred to as "presentation planning".

Part hierarchies permit a strategy to decide what to show and how to avoid information overload: don't show all the parts of a requested object. If a simple display is expected, just show an integral object. If a more informative display is expected, then show the integral object with its parts. More generally, control the complexity of a display by selecting the number of levels of the part hierarchy that are shown on the screen.

The Class Hierarchy

In our theory a non-tangled class hierarchy is used for standard downward inheritance. However, we also supply a limited upward inheritance facility. We find justification for this in the psychological research on categorization. The cognitive science literature reports three different approaches to categorization the *classical* approach, the *prototype* approach and the *exemplar* approach. The classical approach has been but totally rejected from a cognitive point of view. It requires that every member of a class is described by necessary and sufficient conditions.

The prototype view as developed by E. Rosch [Rosch, 1978] describes a "prototype" as a summary description of all the members of a class. The third theory of categorization is the exemplar view. The exemplar view differs from prototype theory in the following way. The summary description used by prototype theory is not necessarily identical to any existing member of the category. Exemplar theory on the other hand postulates the use of one or more stored real exemplars of the category, in other words **no** summary description exists.

The exemplar view of categorization permits us to think in terms of upward inheritance from an individual to a class, because if we do not assume a summary description we may not associate attributes with it, and then the only source to derive inherited attributes from are other exemplars. This implies that it must be possible to inherit attributes from one exemplar upwards to a class and then back downwards to another exemplar.

For example, a knowledge base in our system might contain an object with no specified form that belongs to a class hierarchy. Classical downward inheritance would search up in the hierarchy until at some higher level a form is encountered. However, it might happen that no form is found anywhere in the hierarchy. In our interpretation of the exemplar theory it is valid to do a *"down"* search in the hierarchy for an object that belongs to the same class as the current focus object, and to inherit an existing form with *"up-and-down inheritance"* for it.

The idea of up-inheritance is not popular in AI. It is either ignored or explicitly prohibited. For instance, knowledge representation of the NETL style [Fahlman, 1979] which is based on marker passing, prohibits the idea of inheritance according to an up-and-down-movement because if one would permit markers to move up and down in the network the whole network would eventually be marked.

One is tempted to interpret up-and-down inheritance by considering the first step (the up-inheritance) as a form of generalization or inductive learning. However, this is not what we have in mind, because the representation of the class itself is *"not"* changed by a step of up-and-down inheritance. If a class should have many members only one of which has a form, and if this form should be changed after one application of up-and-down inheritance, then the second application of this inheritance rule will supply the new form, not the old form. Were we talking about a step of generalization, then the class would preserve the form after the first use of up-inheritance.

Are we then making a decision for the universal use of exemplar inheritance and against prototype theory? Clearly this is not our intention, because up-and-down inheritance is *"only"* used when no sufficient information is associated with the classes used for inheritance *i.e.*, when our version of a summary description fails. We do not eliminate the use of a summary description!

For practical purposes we have limited the use of up-and-down inheritance in two ways. Up-search is done only from the lowest level, the level of the individuals, to the level immediately above it, *i.e.*, to the lowest level of classes. If there is no other member in this class, or if the other members do not carry the desired information, then up-and-down inheritance fails. One can argue that this does not make complete use of the class hierarchy, but it seems like a reasonable compromise, because humans use hierarchies that are flat and bushy. Rosenfeld has even argued that it is not necessary to view operations on hierarchies as recursive to an arbitrary depth, because this constitutes an unnecessary effort if one has only a flat hierarchy.

Secondly, up-and-down inheritance is used only for information that is urgently needed, and not as the default case. In a graphics system the one item of information that is obviously needed most is the form of an object, for which no "reasonable defaults" can be supplied. We will now formally define up-and-down inheritance which we also refer to as exemplar inheritance.

Definition: Exemplar Inheritance. If an individual is missing information about an important property, and this property cannot be derived by inheritance from a superclass of the individual, then the property may be inherited from any of the other members of the *immediate* superclass of the individual.

In our domain only "forms" are considered important, and we have therefore decided not to represent the fact that a property is important by an explicit assertion.

It is not yet clear what happens when several members of a class offer different properties for upward inheritance. In such a case a combined strategy of majority and recency may be used.

Reasoning

The major reason for introducing the notion of graphical deep knowledge as separate from graphical knowledge has been the interest in doing reasoning about graphical structures. The first step of making a corpus of representations accessible to logic based reasoning is

to transform it into a well formed declarative format with a defined syntax and semantics. It has been the approach of this investigation to limit the procedural representations which at some point are not avoidable in graphics to a small area, namely to iconic primitives. All conceptual relations between these iconic primitives are represented declaratively.

The second step is to formally define reasoning patterns. SNePS provides two different facilities for doing so, a system of rules and a system for defining paths. Although the rules that can be defined are very powerful and permit quantification as well as the use of non standard connectives we have chosen to concentrate in our implementation on the use of paths which are more efficient.

Path based inference in SNePS assumes that one has a node of a well specified category available (typically an "object") and follows the arcs that are pointing to this node backwards until one hits a node describing unknown and interesting information (for instance a "form" or one coordinate of a position). The well specified case frames of GDK assure that if the required information exists at all in the network, then it will be reachable by a well defined path.

Maintenance Interface

The use of the TINA program as a graphics interface of the VMES project is described in this section. The VMES system consists of a maintenance reasoner and a graphics interface. The graphics interface is an application of an older version[3] of the TINA program. The task of the maintenance reasoner is to identify a faulty component in a given device, usually a circuit board. The maintenance reasoner and the display program share a knowledge base realized as a SNePS network.

During the process of identifying a faulty component in a device, the maintenance reasoner repeatedly updates the shared knowledge base. It categorizes components as being in a "default state", being in a state of violated expectation, being recognized faulty or being suspected to be faulty. Information about any of these states is asserted in the network, using the attribute case frame described earlier on. Whenever the maintenance reasoner wants to express changes in its state of knowledge about the analyzed device, it executes a call to TINA. TINA presents the current state of the maintenance process to the user. This is done by mapping attributes into signal colors (red = faulty, blue = default, green = suspect, magenta = violated expectation).

Typically a device will be displayed completely blue in the beginning. After finding a violated expectation, for instance a port that has a wrong voltage value, this port will receive an attribute "violated expectation". The device will now be blue, except for the port in question which will be magenta. Finally, after several steps of reasoning and redisplay, the device will be shown in blue with the faulty component(s) in red.

The procedural interface between maintenance reasoner and display program consists of the TINA function only! All other communication is done through the shared knowledge base that both parts of the program have access to. Our experience with this type of

[3]Based on a VAX 11/780 and a GIGI graphics terminal.

programming has been that it is exceedingly easy to combine two independently developed modules. To our own surprise no integratory debugging was necessary!

3.3.2 Frontend: User Interface to Encode Devices

FRONTEND is a user-friendly piece of software designed to help encode circuit devices in the representation, to perform the diagnosis. In it, basically, the user is:

(1) asked to specify what he wants to encode: physical or logical instantiation-rule, structural-template, or cross-links;

(2) asked questions to elicit the details of the device being represented. The segments of representation code that do not change from device to device are automatically filled in.

(3) led through the various segments of device representation, in order. Thus, any possibility of missing out on some segments of the representation, is totally avoided.

(4) informed to the extent possible, what type of an answer is expected for each question. e.g., D(igital) / A(nalog). Questions are framed as clearly as the topic permits. eg., "What is the physical bit corresponding to the MSB of the remaining 5 bits? ".

(5) offered the ease of answering in as few key-strokes as possible. The emphasis of the package is on cutting to a minimum, the drudgery of the author.

The following are the special features of FRONTEND :

(1) Files are named, opened and closed automatically. When a device is completely coded, the name of the file where the representation of the device can be found, is displayed for convenience.

(2) Each file is documented automatically. Documentation includes the last name of the author, the date of encoding and a brief explanation as to the nature of the contents of the file. This information is patched to the beginning of the file created.

(3) The representation code generated is pretty-printed. This makes reading the code easier. However, this also results in a much larger file than is strictly necessary.

(4) Most of the questions asked not only specify the nature of the answer expected, but also reject unacceptable answers. For example, if a fixed point number is typed in where a non-negative integer was expected, the question is repeated till an integer is entered.

FRONTEND was originally written in FranzLisp on VAX 11/785 and is about 13K in size. It has been now transported to Common Lisp on the TI Explorer 2. It uses about forty functions, most of which have been named in a self-documenting style. The code can be easily changed to incorporate future extensions in representation.

3.4 DIAGNOSTIC REASONING USING MEASUREMENTS

3.4.1 Introduction

In model-based diagnosis, there are three major steps: discrepancy detection, candidate generation and candidate discrimination/confirmation. Discrepancy detection is the process of identifying the discrepancy between the predicted and observed values at certain places, e.g., at the primary outputs of a circuit. After a discrepancy is detected, the failure symptom is used to generate a set of candidates which can potentially explain the observed symptom when some of them are faulty. Here we use the simplest candidate generation procedure which collects components connecting to any violations. Usually, the candidate generation procedure produces more than one candidate for a particular symptom. This calls for the need to discriminate or confirm candidates as well as the need to select the most likely candidate to test first. The latter motivates the initial ordering of candidates based on structure and symptom information. Simple probes at the inputs and outputs of a candidate are used to determine if it is actually faulty under the current test.

Measurements are used not only to determine the status of a candidate but also to determine the relative fault possibilities of other candidates. As more measurement information becomes available, some candidates become more likely [illegible] to be faulty. This intuition is the [illegible] and [illegible] based. New measurements are [illegible] making the [illegible] more violations [illegible] assumptions are made [illegible] constant so that measured [illegible] state of a circuit is assumed to be [illegible] and the [illegible] of inputs [illegible] outputs.

3.4.2 Candidate Ordering, Reordering and Elimination

This section summarizes the [illegible] candidate ordering [illegible]. A complete account for these topics [illegible]. As mentioned earlier, when more than one candidate is produced by the candidate generation step, it is desirable for a diagnostic system to be able to focus on the most likely candidate first so that the faulty parts can be located earlier. Intuitively, we have the following two heuristics: (1) a submodule is more likely to be faulty if it is connected to more bad primary outputs and (2) a submodule is less likely to be faulty if it is connected to more good primary outputs. Initially, candidates are ordered according to their relationships with incorrect and correct primary outputs as described.

While the initial candidate ordering looks promising, there is no guarantee that actual

3.4 DIAGNOSTIC REASONING USING MEASUREMENTS

3.4.1 Introduction

In model-based diagnosis, there are three major steps: discrepancy detection, candidate generation and candidate discrimination/confirmation. Discrepancy detection is the process of identifying the discrepancy between the predicted and observed values at certain places, e.g., at the primary outputs of a circuit. After a discrepancy is detected, the failure symptom is used to generate a set of candidates which can potentially explain the observed symptom when some of them are faulty. Here we use the simplest candidate generation procedure which collects components connecting to any violations. Usually, the candidate generation procedure produces more than one candidate for a particular symptom. This calls for the need to discriminate or confirm candidates as well as the need to select the most likely candidate to test first. The latter motivates the initial ordering of candidates based on structure and symptom information. Simple probes at the inputs and outputs of a candidate are used to determine if it is actually faulty under the current test.

Measurements are used not only to determine the status of a candidate but also to determine the relative fault possibilities of other candidates. As more measurement information becomes available, some candidates become more likely, less likely or impossible to be faulty. This intuition is the foundation on which candidate reordering and elimination are based. New measurements are also used to determine when a diagnosis can be terminated without making the single fault assumption. A diagnosis can be terminated when there are no more violations which cannot be explained by the faults found so far.

Some assumptions are made throughout the discussion. We assume the faults are non-intermittent so that measured values are independent of time. Probing at any intermediate points of a circuit is assumed to be always possible. We also assume there is no bridge fault and the directions of inputs and outputs are correct. However, single fault assumption is not required.

3.4.2 Candidate Ordering, Reordering and Elimination

This section summarizes the principles of candidate ordering, reordering and elimination. A complete account for these topics can be found in [Chen and Srihari, 1989]. As mentioned earlier, when more than one candidate is produced in the candidate generation step, it is desirable for a diagnostic system to be able to focus on the most likely candidate first so that the faulty parts can be located earlier. Intuitively, we have the following two heuristics: (1) a submodule is more likely to be faulty if it is connected to more bad primary outputs and (2) a submodule is less likely to be faulty if it is connected to more good primary outputs. Initially, candidates are ordered according to their relationships with incorrect and correct primary outputs as described.

While the initial candidate ordering looks promising, there is no guarantee that actual

```
procedure diagnose (CL: an ordered candidate list)
  while CL is not empty do
    Instantiate the first candidate at its level-1 abstraction
    Measure its inputs and outputs
    if it has violated outputs then
      if its corresponding physical object is at IML then
        Issue repair order for the physical object
      else
        Instantiate it at its level-2 abstraction
        Generate and order suspected components of it using its structural description
        Call diagnose on the ordered suspected components
      endif
    else
      Claim that the current candidate is intact
    endif
    Eliminate candidates
    Reorder the remaining candidates
    Propagate measurements to update predications, violations and CL
  endwhile
  Report findings
endprocedure
```

Figure 3.4.1: Control structure of VMES

faulty components are ordered in the front of the candidate list. In fact, for any device and good/bad output pattern, it is not difficult to come up with a counterexample on which our method does poorly in the sense that the actual faulty component is put at the last few places in initial ordering. To fix this problem, we reorder or eliminate candidates whenever some intermediate values are measured.

After the inputs of current candidate are measured, some candidates become more likely to be faulty than others. Obviously, candidates connecting to inconsistent inputs are more likely to be faulty than those to consistent ones. Therefore, candidates connected to its incorrect inputs are shoved to the front of candidate list and candidates connected to correct inputs but not to incorrect inputs are shoved to the tail.

In addition, some candidates may become impossible to be faulty after new measurements are known. As a result, candidates that no longer have a path to any violations can be removed from the candidate list.

3.4.3 Termination of Diagnosis

The control structure of VMES is shown in Figure 3.4.1. It starts from the top level of

the structural hierarchy of the diagnosed device by instantiating the device at its level-1 abstraction. It then tries to find the device's output ports that violate their expectations (i.e., output ports that have different observed values from expected ones). A candidate is claimed to be intact (with respect to the current inputs) if it has no violated outputs.

After detecting violated outputs of the current candidate, the system determines if it is necessary to examine the components of the candidate based on the idea of "intended maintenance level" (IML). The candidate is declared faulty and a repair plan is formed for its corresponding physical object if the physical object is at the intended maintenance level selected by the user at the beginning of the diagnosis session.

Otherwise, the candidate is instantiated at its level-2 abstraction. The structural description is then used to find, at the next lower hierarchical level, a subset of its components which might be responsible for the violated outputs of the current candidate. These suspected components are ordered according to the initial ordering criteria. This diagnosis process is then recursively called for the new ordered components.

After the current candidate is checked, its input and output measurements are used to reorder and eliminate some of the remaining candidates as described earlier. Also the measurements are propagated toward the primary outputs of its super-part at the next higher hierarchical level so that the predicted values are up-to-date. As a result, the violations and candidate list are updated too. The diagnosis terminates when there are no more candidates, i.e., when there are no more unexplained violations. This strategy enables the system to diagnose multiple faults without the explicit enumeration of all multiple fault hypotheses and yields good performance in terms of the number of checked components as discussed in the next section.

3.4.4 Analysis

To show that candidate reordering and elimination really help shorten the length of a diagnosis, we compute the average number of components that have to be checked before the culprit is found, under single fault assumption (SFA). Note that under SFA, shoving candidates connected to incorrect inputs of current candidate to the front of candidate list is equivalent to removing other candidates as far as the length of diagnosis is concerned. The culprit is guaranteed to be among those being shoved to front since they contribute to some violations while others don't. Throughout the analysis, the probability distribution is assumed to be uniform. For example, each candidate has equal failure rate and a candidate has a probability of 0.5 of being non-error-propagating if it is not faulty.

Let $len(n, k)$ denote the average length of a diagnosis (number of components checked) when there are n components and k faults in the diagnosed device. Then $len(n, 1)$, the average length of a diagnosis under SFA, is given by the following recurrence relation:

$$\begin{aligned} len(n,1) &= \frac{1}{n} \cdot 1 \\ &\quad + \frac{n-1}{n} \cdot \frac{1}{2} \cdot \left[1 + \sum_{i=1}^{n-1} \frac{1}{n-1} \cdot len(i,1)\right] \end{aligned}$$

$$+\frac{n-1}{n}\cdot\frac{1}{2}\cdot[1+\sum_{i=1}^{n-1}\frac{1}{n-1}\cdot len(i,1)].$$

The first term represents the case that the first candidate is faulty (with probability $1/n$) and only one component is checked. If the first candidate is intact (with probability $(n-1)/n$) and non-error-propagating (with conditional probability 0.5), 0 to $n-2$ candidates may be eliminated (i.e., 1 to $n-1$ candidates are left) and each case has an equal probability of $1/(n-1)$. The average number of checked components for this case is computed by the second term. Similarly, if the first candidate is intact (probability $= (n-1)/n$) and error-propagating (probability $= 0.5$), 1 to $n-1$ candidates may be shoved to the front of candidate list. This is described by the last term. The above expression simplifies to

$$\begin{aligned} len(n,1) &= \frac{1}{n} + len(n-1,1) \\ &= \frac{1}{n} + \frac{1}{n-1} + len(n-2,1) \\ &\vdots \\ &= \frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{2} + 1 \\ &= \sum_{i=1}^{n}\frac{1}{i} \\ &= \log n.^{1} \end{aligned}$$

This is much better than random or sequential examination whose expected length is $\frac{1}{2}n$.

Our diagnostic reasoning procedure does not make SFA, and a closer examination of the above analysis reveals that SFA is not necessary in the analysis either. The analysis applies to the average number of checked components to find the first fault in a device with n components. Therefore, $len(n,k)$ can be generalized to denote the average number of checked components to find the first k faults in a device with n components and at least k faults.

Let $p_i(\geq 0)$ be the probability of the first fault being found at the ith time $(i = 1,\ldots,n)$. Then, $\sum_{i=1}^{n} p_i = 1$ by the definition of probability and $\sum_{i=1}^{n} p_i \cdot i = len(n,1) = \log n$ according to the analysis for single fault cases. Now $len(n,2)$ can be described as follows:

$$len(n,2) = \sum_{i=1}^{n} p_i \cdot [i + len(n-i,1)],$$

where $[i + len(n-i,1)]$ is the average checked components when the first fault is found at the ith attempt and p_i is the probability of this case. Since $len(i,1) = \log i$,

$$\begin{aligned} len(n,2) &= \sum_{i=1}^{n} p_i \cdot i + \sum_{i=1}^{n} p_i \cdot \log(n-i) \\ &= \log n + \log[\prod_{i=1}^{n}(n-i)^{p_i}]. \end{aligned}$$

[1] The base of all logarithm functions in this section is the natural number.

Number of faults	Number of candidates	Last fault position	Length of diagnosis
1	11.55	3.27	1.93
2	12.22	8.08	3.46

Table 3.4.1: Average results of 100 trials

By geometrical inequality, $\prod_{i=1}^{n}(n-i)^{p_i} \leq \sum_{i=1}^{n} p_i \cdot (n-i)$. Since the function log is monotonically increasing,

$$\begin{aligned} len(n,2) &\leq \log n + \log[\sum_{i=1}^{n} p_i \cdot (n-i)] \\ &= \log n + \log[n - \sum_{i=1}^{n} p_i \cdot i] \\ &= \log n + \log(n - \log n) \\ &\leq 2\log n. \end{aligned}$$

By mathematical induction, we have

$$len(n,k) \leq k \log n,$$

where $1 \leq k \leq n$. This means that the length of diagnosis, when the number of faults is relatively small (which is true for real diagnostic problems), grows logarithmically with respect to the number of components.

3.4.5 Simulation Results

The performance of candidate ordering, reordering and elimination is analyzed by simulation on a 4-bit comparator (see Figure 3.4.2) with 15 components. Each time some randomly selected components are made faulty by complementing some arbitrarily chosen outputs of those components. The number of initial candidates, the last position of those components in the initial candidate ordering, and the actual number of checked components are recorded for analysis.

The average results of 100 trials for both one and two fault cases are summarized in Table 3.4.1. The third column indicates that initial candidate ordering performs well for the first fault but not necessarily for the second fault. The last column shows that the average length of diagnosis for two faults, 3.46, is less than twice of that for single fault, 1.93. This is consistent with the asymptotic analysis presented in the previous section.

3.4.6 Discussion

We have presented a model-based diagnostic reasoning procedure which uses measurements to locate faults, shorten candidate list and terminate a diagnosis. Probing is not the only

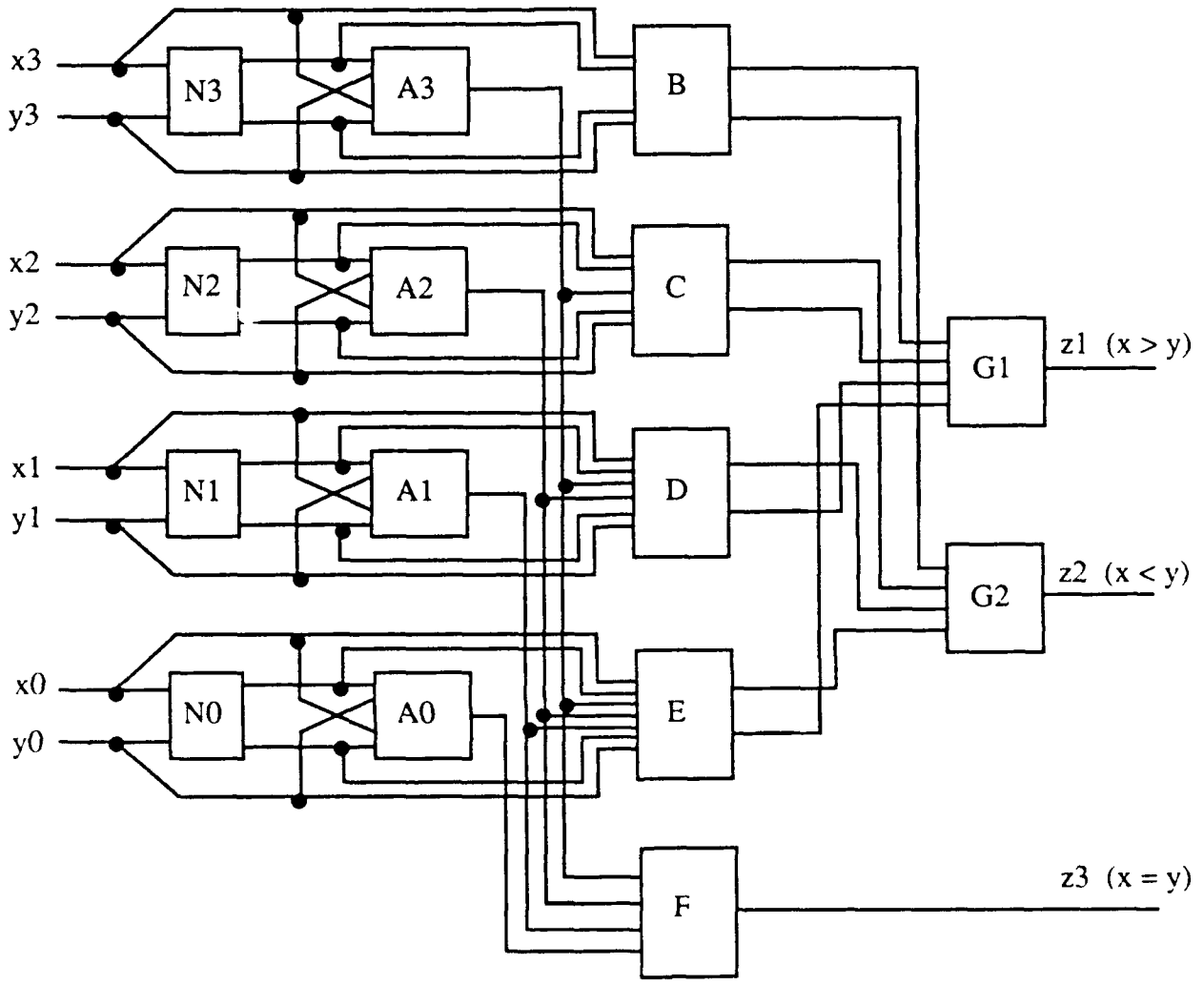


Figure 3.4.2: Logical diagram of a 4-bit comparator

way to obtain additional information. Board swapping, which is commonly used by human technician, can also be used for those tasks.

Instead of probing the inputs and outputs of the first candidate, we can replace the board containing it with a known good board. Then, we apply the same test to the primary inputs and compare the output vector after the replacement with the old one. If the two vectors are identical then the original board was not faulty and the diagnosis proceeds with the remaining candidates which are not contained in the board. If they are different and the new one is correct then the original board was faulty and the diagnosis can be terminated. Otherwise, the original board was faulty and there are still unknown faults. The diagnosis continues on the remaining candidates reordered according to the new symptom.

Applying a different test to the device is yet another approach. Shirley and Davis developed a method for generating tests to discriminate candidates using hierarchical models and symptom information [Shirley and Davis, 1983]. It generalizes traditional path sensitization by using hierarchical and symptom information to avoid or "tunnel under" candidates. One restriction of their method is the single fault assumption. An extension of the method by relaxing the single fault restriction is currently under investigation

3.5 REPRESENTATION AND DIAGNOSIS OF SEQUENTIAL CIRCUITS

3.5.1 Introduction

Constraint based representation of structure and behavior has been traditionally used to localize faults in combinational circuits [Davis, 1983]. This scheme was extended to represent sequential circuits, with the use of layers of temporal granularity and a vocabulary of signals appropriate to the circuit [Hamscher and Davis, 1984]. Expectation violation has been used for candidate generation in combinational circuits. The same procedure, when applied to sequential circuits, was found to be indiscriminate. Single stepping was suggested as a divide and conquer strategy to localize faults in sequential circuits. Structural detail was however proposed to make candidate generator discriminating [Hamscher and Davis, 1984].

Here, we present the details of using structural detail to diagnose sequential devices. We outline candidate generation based on electrical behavior, using fault characteristics which conveniently express structural details. We detail representation and handling of sequential circuits, its unique problems and theoretical / technical solutions. Further, we sketch the diagnostic steps necessary for sequential devices, over and above those used for combinational devices. Finally, we talk about assumptions, and how they should be relaxed in a multiple-symptom case, so as to make diagnosis efficient and correct.

3.5.2 Representation of Sequential Circuits

Hierarchical representation of circuits aids proper focusing of diagnosis. It makes details available on an as-necessary basis, so that there is neither glut nor dearth of information during diagnosis. Therefore, the structure of circuits has been represented in layers of hierarchy.

Sequential circuits are admittedly more complex than combinational circuits because they have an added time dimension. Their behavior may vary with time because of in-built memory. To conveniently model sequential circuits, this time factor must be taken into account. Therefore, we proceeded to propose a temporal hierarchy for sequential circuits.

The following algorithm lays the rules for temporal hierarchy among circuits. It starts with the most basic blocks and builds layer by layer, up to the most complex circuits.

(1) Basic gates are represented at two levels: the gate delay level which is necessary for analysis of faulty devices causing racing conditions in circuits; and at the input-output relation level. This maybe truth-table or boolean expression. For most purposes the second representation will be adequate. Moreover, since VMES does not deal with parametric faults, the first representation has been ignored in the current VMES system.

(2) Flip-flops are represented at two levels: the clock period level and the overall behavior level. The overall level may be transition diagram, state table or just a function.

(3) Any complex unit such as a chip, which is at the basic maintenance level is represented as in (1) or (2) depending on whether it is sequential or combinational.

(4) Complex modules such as boards are represented at two levels, if possible: at the level lowest among the highest levels of representation of the immediate sub-modules; and at the overall behavior level of the board.

The different levels of time representation chosen are however expressible as integral multiples of the more basic levels. Also note that the two levels of representation suggested may be the same for some devices. This scheme is applicable to most general cases of synchronous circuits.

3.5.3 Handling Feedback in Sequential Circuits

Sequential circuits are made up of sequential and combinational components or subdevices. These components interact closely to constitute the behavior of the whole circuit. By virtue of this close interaction, subdevices of a sequential circuit are better addressed from the perspective of the encompassing superdevice. As we will see along this section, this applies to the representations of function and fault characteristics and handling of states of components in a sequential circuit. The terms 'circuit' and 'superdevice' are used interchangeably in the following discussion, as are the terms 'component' and 'subdevice'.

Sequential circuits have feedback, also termed memory. Memory is dealt with as the state of the device. Output of a sequential device depends not only on the inputs but also on its state. The state of a device is in turn, a function of its outputs in the previous clock cycle. Hence, representation of a sequential component should provide for the storage of the state of the device. This storage variable should be initialized at the beginning, and re-assigned after every clock cycle.

Initialization of a circuit involves initializing all the sequential subdevices constituting the circuit. Initialization of a component consists of setting its state variable(s) to a value. This value is either a fixed number or a function of some inputs of the superdevice. Initialization routines for subdevices are stored with the superdevice and are executed upon entry into the superdevice.

Functions of sequential components are expressed at the lower level of temporal hierarchy of the encompassing superdevice. For instance, if 8 bit adder is a subdevice in a sequential multiplier, and the temporal hierarchy of the sequential multiplier consists of (1) functional level : out1 = in1 * in2 and (2) clock cycle level, the function of the 8 bit adder is expressed per clock cycle. During simulation of the superdevice, the functions of the subdevices are executed in topological order as many times as the number of clock cycles in the superdevice. We note that application of function in sequential components involves states besides input and output ports. The value of state may be used as an input or the state variable maybe treated as on output for the component function.

Due to the necessity of feedback, sequential circuits usually have closed loops in their topology. However, to carry out simulation and reasoning step by step, (at the relevant temporal unit) these loops must be severed appropriately to provide pseudo-input and

pseudo-output ports. In other words, it is necessary to recognize the subdevice(s) that store the state of the superdevice, and count the inputs and outputs of these subdevices as pseudo-outputs and pseudo-inputs of the superdevice respectively. *Loopbreaks* are the pseudo-inputs of the superdevice that carry the previous state of the superdevice for purposes of function application. *Loopbreak-heads* are the subdevices whose outputs are the pseudo-inputs of the superdevice. During simulation of the superdevice behavior, the following steps are carried out in order:

(1) The superdevice and its relevant sequential subdevices are initialized

(2) First time around, starting with primary inputs of the superdevice and outputs of the *loopbreak-heads*, functions of every subdevice is executed in topological order. (This order may not be strict and may involve backtracking) The topological order stops either at primary outputs or after *loopbreak-heads*.

(3) On subsequent rounds, one of two procedures may be followed: If the superdevice takes in a fresh set of inputs on every round, procedure (2) is repeated. If however, the superdevice works on only one set of inputs (i.e., those taken in on the first round) procedure (2) is carried out with the outputs of the subdevices connected to the primary inputs of the superdevice substituting for the primary inputs of the superdevice. Step (3) is repeated for as many times as the function of the superdevice warrants. For instance, for the 4 bit sequential multiplier that takes 4 clock cycles to calculate the product, each of the above rounds may be taken as one clock cycle. Thus, representation and simulation of sequential circuits is considerably different from that of combinational circuits because of the inherent feedback.

3.5.4 Candidate Generation based on Electrical Behavior

Sequential circuit diagnosis is admittedly hard. Therefore, every opportunity to narrow down the list of possibly faulty devices should be exploited to the maximum. With this goal in mind, we propose candidate generation based on electrical behavior.

Candidate generation based on electrical behavior rests on the following observation: Not all the inputs of a component will be responsible for some observed wrong value at its output; Further, many of these inputs that are not responsible for the faulty output can be identified and eliminated from further consideration with full certainty. The knowledge, which inputs are not responsible, or in other words, which inputs may be responsible for an observed wrong value at the output of a component, is expressed as the fault characteristics of the component. For complex devices, i.e., superdevices, fault characteristics also capture the list of subdevices that are possibly faulty, given some faulty values at the outputs of the superdevices. In short, fault characteristics are indicators of culpability. For example, fault characteristics for an AND gate are as follows. In the table, output values are measured, input values are expected and the last column specifies the inputs to suspect and to propagate backwards from.

AND gate Fault Characteristics			
OUT1	IN1	IN2	SUSPECT / PROPAGATE
0	1	1	either in1 or in2 or both
1	1	0	input supposed to be 0 : in2
1	0	1	input supposed to be 0 : in1
1	0	0	none if single fault assumption unless inputs tied

Following are some note-worthy observations about fault characteristics:

(1) Knowledge of the value of the faulty output is necessary to apply fault characteristics. The characteristics vary with the faulty value. Moreover, characteristics are particular to the output of the component, and change from one output to another, unless the function (and in cases, the internal structure) of the outputs are the same.

(2) Depending on the nature of the device, fault characteristics are helpful to varying degrees. For example, for the AND gate, they are helpful in all cases except when both inputs are 1. However, for the exclusive OR gate, as seen below, they are not of much help except to point out that only one input could be wrong in any given case.

Ex-OR gate Fault Characteristics			
OUT1	IN1	IN2	SUSPECT / PROPAGATE
1	0	0	either in1 -> 1 or in2 -> 1
1	1	1	either in1 -> 0 or in2 -> 0
0	0	1	either in1 -> 1 or in2 -> 0
0	1	0	either in1 -> 0 or in2 -> 1

(3) For all simple devices, fault characteristics could be pre-computed and pre-compiled. With this information readily available, diagnosis can be sped up considerably.

(4) Since sequential circuits have closely interacting components, fault characteristics can less apply to individual components than to complex superdevices. The fault characteristics of a superdevice specify which subdevices to suspect, given some violation at the outputs of the superdevice. Again, these characteristics are expressed at the higher level of temporal representation of the superdevice. The lower level of representation is typically utilized for candidate elimination. For example, for the 4 bit sequential multiplier, fault characteristics are expressed in terms of the possible products that may appear at the outputs of the multiplier. However, user is asked for measurements at the clock cycle level, so that measured and computed values can be compared for candidate elimination.

(5) Fault Characteristics capture the internal structure of devices, especially for sequential devices. They introduce structural information into sequential circuit diagnosis, thereby making the task more feasible. Therefore, not only the function but also the internal structure of a component is required to generate fault characteristics for the component.

Fault characteristics can be easily formalized for only the simpler devices that reside at the lowest level of structural hierarchy. For more complex devices, it is hard, even infeasible and unnecessary to have pre-computed fault characteristics. Instead, fault characteristics for the device, for the observed faulty output, is computed on the run from the characteristics of the subdevices, and as part of the diagnostic procedure. This procedure is as

follows:

```
FOR each faulty output of the device
    get all the devices connected to it
    FOR each device connected to the output
        get its fault characteristics
        find all the inputs responsible for the faulty output of the device
        recurse on each of the inputs
    END-FOR
END-FOR
```

The fault characteristics so computed simply constitute the list of candidates that need to be checked. The above procedure could be tuned to further narrow down this list by incorporating forward reasoning in case of single fault assumption. In essence, this means that when a new subdevice is added to the list, the algorithm reasons forward from the subdevice towards the primary outputs of the superdevice. If this reasoning ends at any good output of the superdevice, then the subdevice can be eliminated from the list right away. This procedure is based on the observation that a bad subdevice output cannot contribute to a good superdevice output when single fault assumption is made. This has not been built into the system VMES yet.

Candidate generation based on electrical behavior is significantly different from candidate generation based on topology [Chen and Srihari, 1989]. In the topological procedure, only the knowledge of whether an output is faulty or not suffices for candidate generation. However, in the electrical procedure, we will also need the details as to how the output is faulty (i.e., measured values). In diagnosis, it is reasonable to assume the availability of this information. Topological procedure is heuristic, whereas electrical procedure is algorithmic. The advantages and disadvantages of heuristics versus algorithms mostly apply to the comparison of the two procedures as well. Whereas topological procedure works fine without any knowledge of the functions of the subdevices in the circuit, electrical behavior cannot do without the information. Since electrical procedure utilizes more information about the circuit in question, it can be expected to perform at least as well and possibly better than the topological procedure. However, the associated cost (in terms of extracting fault behavior, and using electrical behavior for candidate generation) is also higher.

3.5.5 Diagnosis of Sequential Circuits

The following control structure of VMES was adequate to diagnose combinational circuits:

```
REPEAT
    Simulate the circuit at the next structural level
    Single step once down the structural hierarchy
    Generate candidates by constraint elimination
    Eliminate candidates by asking for more information
UNTIL circuit diagnosed or basic level of structural hierarchy reached.
```

However, Sequential circuits have temporal complexity in addition to the structural complexity found in combinational circuits. Therefore, temporal hierarchy was proposed earlier for sequential circuits. The control structure had to be suitably modified to handle this hierarchy and exploit its advantages. The new control structure reads as follows:

REPEAT
    Initialize the subdevices with states (at the next lower structural level)
    Simulate the superdevice at its next lower temporal level, if possible.
    Apply fault characteristics to narrow down the subdevice suspect-list
    Single step once through temporal hierarchy (if possible);
    Single step once through structural hierarchy;
    Eliminate candidates by asking for more information;
UNTIL fault diagnosed or basic level of structural and temporal hierarchy reached.

In the above algorithm, note that: (1) Representation scheme and control strategy are designed to be compatible.

(2) Stepping down the temporal hierarchy may not always be possible, because, a device may have only one level of temporal representation.

(3) Fault Characteristics are applied at the superdevice level, and at the superdevice's higher temporal level.

During candidate elimination, assuming complete visibility, measured values are asked at the ports of the candidates. These measured values are compared with expected values that are computed during the simulation stage of the device. Finally, the following algorithm is used to diagnose / eliminate candidates from further consideration:

For the device at hand:
IF output not as expected
    IF output cannot be explained by inputs
        IF device has subdevices
            generate suspects among subdevices
            and recurse on each of them.
        ELSE declare device to be faulty
    ELSE declare device to be correct
ELSE declare device to be correct

For example, Suppose a 4 bit sequential multiplier outputs 49 on inputs 6 and 8. (see Fig. 3.5.1) The algorithm simulates the multiplier, at its next lower structural and temporal level, i.e., at subdevice level, for each clock cycle. Next, fault characteristics are applied on the superdevice, i.e., the multiplier to narrow down the list of subdevice suspects. Now, the algorithm steps down the structural hierarchy to the subdevice level (adder, 4 bit register, driver etc.) and down the temporal hierarchy to clock cycles. The user is asked for the values of the suspect subdevices during different clock cycles. These values are compared with the values computed during simulation earlier, to come up with a diagnosis.

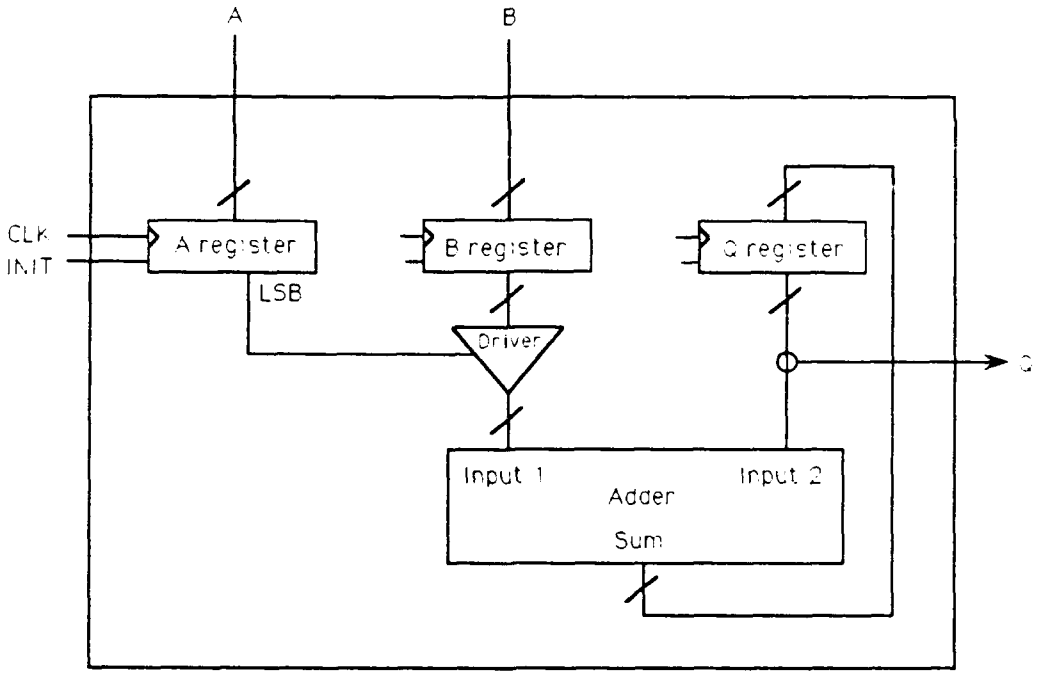


Figure 3.5.1: 4 Bit Sequential Multiplier

3.5.6 Assumptions and their Relaxation

Assumptions are technical conveniences used to make diagnosis easier and faster at the expense of completeness. Some of the common assumptions made during diagnosis are :

(1) Single Fault Assumption

(2) Non-Canceling Fault Assumption

(3) Non-intermittent Fault Assumption

However, in real-life diagnosis, these assumptions are simplistic. They could potentially lead to failure of diagnosis and are hence undesirable. Therefore, assumptions are a matter of trade-off between efficiency and completeness. Ideally, the system VMES should carry on with the assumptions until it is infeasible to do so. This way, the system will have best of both worlds.

We outline a procedure below to relax single fault assumption when many sets of symptoms (input-output pairs) are available for diagnosis.

```
FOR every symptom DO
    find the suspect-list
    find the intersection of this list with
    that generated through previous symptoms
    IF intersection is null, relax single-fault assumption
END-FOR
```

IF assumption still holds, use only the intersection set
ELSE use the union set.

Assuming complete state visibility allows us to take two more liberties:

(1) We can relax non-intermittent fault assumption. The inputs and outputs of all subdevices are available during every clock cycle. Further, candidate elimination algorithm outlined earlier checks if measured output is justified by measured inputs. If this check yields false, irrespective of whether the fault is visible in other clock cycles, it is trapped.

(2) We can relax single fault assumption. Since all ports of all subdevices are measurable, faults can be contained and detected by measurement alone. When the measurements of a subdevice satisfy fault criteria of the candidate elimination algorithm, irrespective of whether other subdevices are faulty, the current subdevice can be declared faulty.

It should be noted that the assumption of complete visibility is simplistic. Ideally, the system VMES should be prepared to deal with situations where no values can be measured off some device. It should be able to work with measured values from only a few of the requested clock cycles. One solution would be to embed information about accessibility of ports and devices so that the system steers the diagnosis towards asking only those values that can be measured. Another would be to make the system flexible so that it can work with partial / alternate data. These ideas have not yet been incorporated into the current system.

3.5.7 Conclusion

We have proposed an algorithm to represent complex sequential circuits so as to facilitate diagnosis. Having outlined candidate generation based on electrical behavior, we have extended it to introduce structural information into sequential device diagnosis. With the proposed control structure of structural and temporal single-stepping, we have attempted to exploit the advantages of complete state visibility. Finally, we have produced an algorithm to relax single-fault assumption towards making diagnosis efficient and correct.

3.6 NEW TEST DEVICE

3.6.1 Introduction

Although this project is intended to develop a system that is adaptable to a wide range of devices in the domain of digital circuits, it was essential to consider a device of reasonable complexity to verify the theory and effectiveness of our diagnosis methodology. Upon recommendation by RADC, a Heathkit Printer Buffer Board was selected as a test bed.

The following criteria were used to select this device:

- combinational circuitry
- sequential circuitry
- analog circuitry
- a variety of component types
- many logical components (> 50)
- complexity sufficient to subdivide into numerous levels of hierarchy

The printer buffer board was assembled locally. Since we built the printer buffer from a kit we know more about its physical construction than we have about any of the other test devices. Due to the fact that we actually have the physical device, it was anticipated that we can cause faults in the device in a more natural way. For instance, we can inject faults by removing or "damaging" components in a number of ways. This is in contrast to earlier situations where there was no physical device to check the diagnosis on, and faults were created in the representation by software.

3.6.2 Physical Description

Model SK-203 printer buffer is enclosed in a sheet metal housing with the front and rear panels exposing the controls and fixtures. On the front panel are 11 pushbutton switches, three 7-segment LED displays and an LED above the "Swap" pushbutton. On the rear panel are a rocker switch, a power jack, two male DB 9 serial port connectors, and two female DB 25 parallel port connectors. Also, there is an external transformer that plugs into a 120 VAC wall receptacle and outputs 8 VAC, 1 ampere to a plug that mates with the power jack on the rear panel.

Within the enclosure are two circuit boards connected by two 20-pin right angle plugs. The smaller circuit board mounts all of the components that are visible in the front panel of the enclosure. The main circuit board mounts the rest of the components of the printer buffer. This includes a 64180 CMOS microprocessor, eight 64K $\times$ 1 bit dynamic RAMs, an 8K ROM, and many logic and decoder IC's. In addition to these digital components there

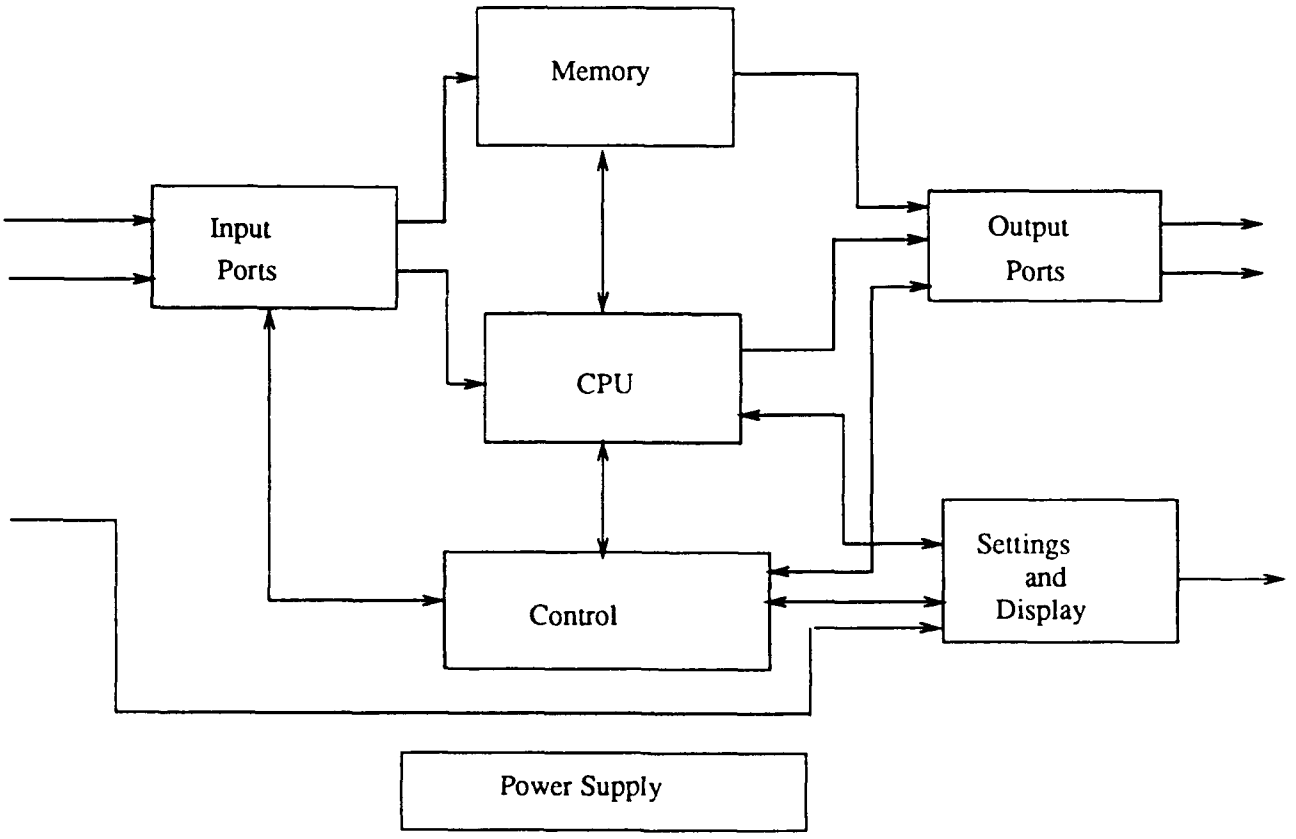


Figure 3.6.1: Logical Units of the Printer Buffer Board at the Top Level

are various resistors, capacitors, diodes, and inductors that give us the analog components of the circuit.

3.6.3 Operational Description

The model SK-203 printer buffer accepts files from one or two computers and sends them to a printer serially or in parallel. By using a combination of the Swap key (pushbutton switch) and the Offline keys all the reasonable configurations can be achieved.

A complete description of the operation of the SK-203 printer buffer is available in the "Heathkit Manual for the PRINTER BUFFER Model SK-203", part number 595-3727-01.

3.6.4 Representation and Diagnosis

The printer buffer board has a natural structural hierarchy. At the top level, it contains a display system, a keyboard input, a control unit, memory and ports. This kind of representation is obtained on the basis of the functionality at the top level. The top level representation is given in Figure 3.6.1. Each of these modules at the top level can in turn be

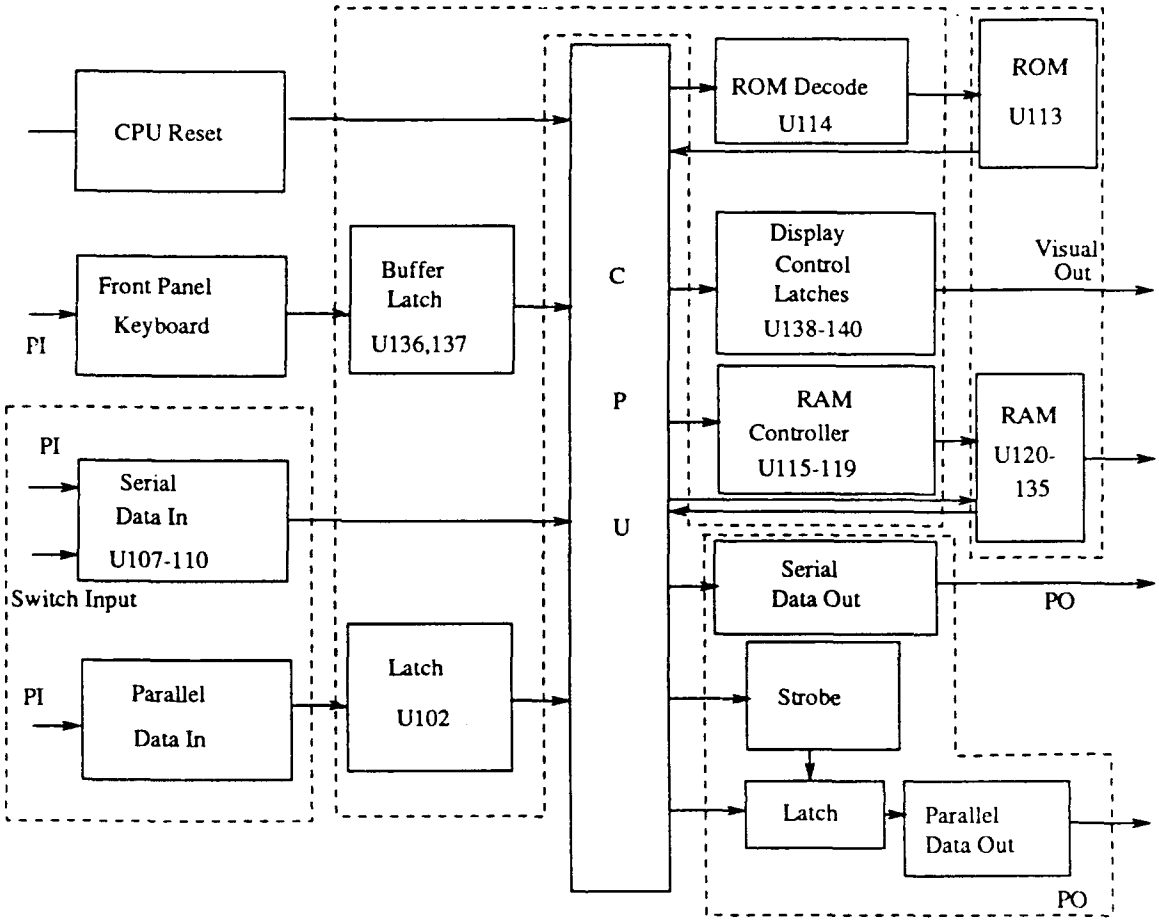


Figure 3.6.2: Intermediate Level Representation

represented at a lower level of abstraction. For example, the control module is composed of ROM controller, RAM controller, Display control and so on. The Display module consists of LED display units, latches to hold the display values and interconnecting wires. We call this level of representation the intermediate level. The intermediate level representation of the printer buffer board is shown in Figure 3.6.2. If one has to diagnose the circuit at the chip level or circuit level, one more level of hierarchical representation is required.

As for the diagnosis of sequential parts of the circuit, the devices have to be put into the following temporal levels: at the gross level of buffer behavior, at the machine instruction level of the controlling CPU, at the clock cycle level and at the gate delay level.

It is worth noting that at the top level, it is rather difficult to represent the functionality of modules in a mathematical equation or a truth table. The level of abstraction is so high that only shallow reasoning is viable. Fortunately, it is possible to capture most of the failures of the modules at the topmost level in shallow rules. In addition, the manufacturers do supply some shallow rules. Some of the shallow rules supplied by Heathkit are:

If the unit is inoperative, check power supply connections;

If the display is random, IC's in Display section have been wrongly installed;

If display fails to indicate free memory, certain jumpers are wrong.

The current VMES has incorporated these rules in its representation in the form of SNePS rules.

Though VMES can diagnose the printer buffer board at its intermediate and lowest levels by means of its model-based reasoning capability, actual diagnosis has not been carried out. The main limitation is the complexity in the representation of functional knowledge at the intermediate level. The main effort at the intermediate level so far has been in identifying the failure modes and the representation of functionality of the various modules at this level. Some of the functional failures the VMES representation can handle are: CPU failure, defective memory and port controller failure. We have analyzed the printer buffer board and attempted to to acquire the functional knowledge of some of the modules at the intermediate level. For example, consider the RAM Controller module consisting of chips U115, U116, U117, U118, U119 and its associated circuitry. Its functionality can be defined by first identifying the inputs and outputs. The inputs to this module are the *Address* and the *Select* lines. Outputs are *Read/Write control* and other *timing* signals. The functional knowledge of this module can be defined as: for a specified address input, a specific chip select signal goes high and memory read or memory write is enabled. This knowledge can be used to test the RAM controller module in the model-based diagnosis approach.

3.6.5 Conclusion

The printer buffer board has served as a good test bed for the VMES research. It has provided an opportunity to test our methodologies and effectiveness of our diagnosis approach. It has revealed the difficulties of acquisition of functional knowledge at intermediate levels of representation and other limitations of migrating theory into practice. More work is required for the acquisition and representation of functional knowledge in model-based diagnosis of circuits of arbitrary complexity.

3.7 A SCHEME FOR SHADOWING GENERAL KNOWLEDGE BY ITS INSTANCES

This section describes new schemes for general AI reasoning systems focusing on the improvement of reasoning performance.

3.7.1 Introduction

The performance of an expert reasoning system is mainly dependent upon how it represents knowledge and how it controls reasoning. Control is needed to resolve knowledge conflicts in which more than one rule is applicable in some problem solving situation. Most expert systems tend to prefer rules of more specific features over rules of more general features as a way of conflict resolution [Sauers, 1988]. This brings the issue of **generality in knowledge** asking how the system distinguishes between more general and less general knowledge? In rule-based systems, the level of generality or specificity of a rule is determined by recognizing **special case** relationship between rules. A relative specificity between rules is defined by McDermott and Forgy [McDermott and Forgy, 1978] : a rule **r1** is more specific than another rule **r2** if (1) the two rules are not equal, (2) **r1** has at least as many antecedent clauses as **r2**, and (3) for each antecedent clause in **r2**, with constant elements $C_1, \cdots, C_n$, there exists a corresponding antecedent in **r1**, with constant elements $C'_1, \cdots, C'_m$, such that $\{C_1, \cdots, C_n\}$ is a subset of $\{C'_1, \cdots, C'_m\}$. According to this definition, a rule A(a,b) & B(a,b) $\Rightarrow$ C(a,b) is treated as more specific than other rules like A(a,b) $\Rightarrow$ C(a,b), $\forall$x {A(a,x) & B(a,x) $\Rightarrow$ C(a,x)}, or $\forall$x,y {A(x,y) & B(x,y) $\Rightarrow$ C(x,y)}. This definition helps us to capture some abstract sense of what is meant by **more general** or **less general** in knowledge.

The concept of knowledge generality may be extended toward the depth of knowledge. In other words, we now intend to classify knowledge in terms of how it qualitatively contributes to solve problems. Some knowledge has the form of `Observation` $\Rightarrow$ `Conclusion` which directly associates inputs with some actions, but does not necessarily provide a reason for the relation between a pair [Chandrasekaran and Mittal, 1983]. We refer this kind of knowledge as **shallow knowledge**. In general, shallow knowledge has no underlying representation of causality or basic physical principles [Hart, 1982], instead it is just a collection of heuristic information such as statistical intuition or past experience of human experts [Reiter, 1987]. Shallow knowledge is usually represented by IF-THEN-like production rules. A typical system which uses shallow knowledge is MYCIN [Shortliffe, 1976]. MYCIN's knowledge base contains a collection of rules describing the relationships between symptoms and disease hypotheses, without specifying the causal links between them [Sembugamoorthy and Chandrasekaran, 1986].

For instance, MYCIN's **steroid rule** [Clancey, 1983] is represented as :

```
IF  (1) the infection which requires therapy is meningitis,
    (2) only circumstantial evidence is available for this case,
```

```
    (3) the type of the infection is bacterial,
    (4) the patient is receiving corticosteroids,
THEN  there is evidence that the organism which might be causing
      the infection are e.coli (.4), klebsiella-pneumoniae (.2),
      or pseudomonas-aeruginosa (.1)
```

On the other hand, **deep knowledge** contains lower-level, causal, and functional information using a qualitative model of the system [Yoon and Hammer, 1988]. There seems to be no strict form for deep knowledge structure, but several alternatives of representing deep knowledge can be summarized in [Chandrasekaran and Mittal, 1983] as : mathematical and simulation models, fundamental physical laws, functional and structural models of a device, causal networks, and sequences of cause effect rules which deduces consequences of events. In medical applications, CASNET [Weiss *et al.*, 1978] is an example system that is based on causal network. A CASNET model consists of observations of a patient and disease categories, which are also components of MYCIN, but it also maintains pathophysiological states that are associated with observations. These states form a network of cause-effect relationships, and patterns of states in the network are related to individual disease classifications. CASNET can explain more deeply the basis on which the final decisions are made about possible diseases.

So far we have discussed various descriptions about the general and specific knowledge distinction and also the deep and shallow knowledge distinction. While the former deals with only the syntactic features by concentrating on the format of the knowledge, the latter is a rather semantical interpretation with vast number of model-theoretic definitions. We will mainly consider the general and specific knowledge distinction since it is easily recognized by the system, but we also expect some of the deep and shallow knowledge representations can be handled with a little modification.

It has been claimed that systems with deep or general knowledge can solve problems of greater complexity than systems with specific knowledge can [Hart, 1982]. But that is not the only requirement for any expert system. We sometimes give more priority to the goodness of the answer, or the reasonable cost to get the answer, rather than the broadness of solving power [Feigenbaum *et al.*, 1971]. It is widely admitted that reasoning by specific knowledge causes less system overload since several intermediate steps are omitted. As a result, specific knowledge will significantly contribute to the good performance of the system. While specificity is needed in a viewpoint of an expert system developer, generality is also needed for a problem-solving researcher.

We propose a systematic way to satisfy both requirements by recognizing generality relations in knowledge. Our approach is divided into 3 different issues : (1) Construction of a multi-level knowledge base by integrating different kinds of knowledge at different levels of generality, (2) Automatic migration of specific knowledge from general knowledge during a reasoning process, and (3) Shadowing general knowledge to select the most specific knowledge when several candidates are applicable.

A multi-level knowledge model is suggested to get benefits from both general knowledge and domain-specific shallow knowledge. It is intended to give the system the power of

generalizability as well as good performance. Mostly deep or general knowledge is domain-independent, which implies that solving a problem by deep knowledge will need some additional steps of inference. For expert systems of real domains that require a large number of rule activations, we can expect that domain independent knowledge leads to serious performance degradation. Our goal is to use domain specific knowledge as far as possible, but the problem is how to relate the general knowledge to its specific counterpart.

3.7.2 Automatic Migration of General to Specific Knowledge

A motivation for the idea of migration comes from the observation of the knowledge derivation mechanism in a deductive reasoning system. The main task of a deductive reasoning system is to derive implicit knowledge from existing knowledge which are known to be true. After derivation, deduced information will be asserted into the knowledge base. In addition to directly derivable knowledge, however, we may get extra information which could be useful for the future reasoning.

To explain this, consider as an example a rule describing the characteristics of a transitive relation between two objects.

`Rule1` : $\forall$`R {transitive(R)` $\Rightarrow$ $\forall$`x,y,z {R(x,y) & R(y,z)` $\Rightarrow$ `R(x,z)}}`

`Rule1` reads: For any relation `R` which has the property of transitivity, if the relation `R` holds between `x` and `y`, and holds between `y` and `z`, then the relation `R` also holds between `x` and `z`. In order to show how the reasoning system automatically deduces new facts, consider a knowledge base contains the following facts as well as `Rule1`.

```
Fact1 :  transitive(supports).
Fact2 :  supports(a,b).
Fact3 :  supports(b,c).
Fact4 :  supports(c,d).
```

Now we want to infer `supports(a,c)` from this knowledge base. A natural deduction derivation [Bibel, 1986] could generate a sequence of inferencing to make `supports(a,c)` true :

`Prop1` : {`transitive(supports)` $\Rightarrow$
$\forall$`x,y,z {supports(x,y) & supports(y,z)` $\Rightarrow$ `supports(x,z)}}`
from Rule1 by Universal Instantiation
with a binding {supports/R}[1]

$\Downarrow$

`Prop2` : $\forall$`x,y,z {supports(x,y) & supports(y,z)` $\Rightarrow$ `supports(x,z)}`
from Prop1 and Fact1 by Modus Ponens

$\Downarrow$

`Prop3` : {`supports(a,b) & supports(b,c)` $\Rightarrow$ `supports(a,c)}`

[1] A binding has a form of {`term1/var1, term2/var2,` $\cdots$ }

from Prop2 by Universal Instantiation
with a binding {a/x, b/y, c/z}

⇓

Prop4 : {supports(a,c)}
from Prop3, Fact2, and Fact3 by AND-introduction
and Modus Ponens

Note that **Prop2** is a useful rule to keep around, and we call it **Rule2**.

Rule2 : $\forall$x,y,z {supports(x,y) & supports(y,z) $\Rightarrow$ supports(x,z)}

Although initially not requested, the derivation of **Rule2** can be justified according to the cognitive aspect of human reasoning. This means that since the relation **supports** becomes known to be transitive in this reasoning, from now on any cognitive agent also should know the nature of transitive relationship for **supports** by **Rule2**. **Rule2** is an instance of **Rule1**, and it is a more specific rule than **Rule1** by the specificity definition introduced in Section 3.7.1. So after the derivation we could assert **Rule2** into the knowledge base as well as **supports(a,c)**. This is an example of a migration of general to specific knowledge during the inference.

As shown by the format of **Rule1**, the concept of migration raises the importance of a scheme of representing a nested rule, or an embedded rule. The specificity level of a migrated rule will be determined by the form of a more general rule represented by nesting with some quantifiers. A semantic interpretation for the rule nesting might say that the embedded definition delivers the intention of a rule builder about the usage of the rule. In the aforementioned example, **Rule2** is migrated during the derivation of **supports(a,c)** according to the form of **Rule1** such that only R is universally quantified at the outmost level. The intention of this rule can be interpreted as finding transitive relationships between objects without having any particular objects in mind.

To explain this more, suppose **Rule1** is represented with different quantifier declarations such as $\textbf{Rule1}_a$ or $\textbf{Rule1}_b$:

$\textbf{Rule1}_a$: $\forall$R,x {transitive(R) $\Rightarrow$ $\forall$y,z {R(x,y) & R(y,z) $\Rightarrow$ R(x,z)}}
$\textbf{Rule1}_b$: $\forall$R,x,y {transitive(R) & R(x,y) $\Rightarrow$ $\forall$z R(y,z) $\Rightarrow$ R(x,z)}}

$\textbf{Rule2}_a$ and $\textbf{Rule2}_b$ might be migrated from $\textbf{Rule1}_a$ and $\textbf{Rule1}_b$, respectively, in the derivation of **supports(a,c)**.

$\textbf{Rule2}_a$: $\forall$y,z {supports(a,y) & supports(y,z) $\Rightarrow$ supports(a,z)}
$\textbf{Rule2}_b$: $\forall$z {supports(b,z) $\Rightarrow$ supports(a,z)}

These rules may also be useful for some particular applications in which the rule builder has some specific objects in mind, or the knowledge base has many entries about the relationship between particular objects. In this example, $\textbf{Rule2}_a$ focuses on the object **a**, and $\textbf{Rule2}_b$ on **a** and **b**. Note that **Rule2**, $\textbf{Rule2}_a$, and $\textbf{Rule2}_b$ have different levels of generality. This implies that different rules at different levels of generality can be migrated from the same kind of rule with just different declarations of universal quantifiers.

```
Fact1 :  transitive(supports)
Fact2 :  supports(a,b)
Fact3 :  supports(b,c)
Fact4 :  supports(c,d)
Fact5 :  supports(a,c)
Rule1 :  ∀R {transitive(R) ⇒ ∀x,y,z {R(x,y) & R(y,z) ⇒ R(x,z)}}
Rule2 :  ∀x,y,z {supports(x,y) & supports(y,z) ⇒ supports(x,z)}
```

Figure 3.7.1: A knowledge base after the derivation of `supports(a,c)`

The mechanism of representing rule nesting is not emphasized in most automated reasoning systems, especially those systems based on resolution and unification. Although the definition of a well-formed formula in the resolution-based system [Robinson, 1965] allows embedded representations, those rules need to be translated into clause form in order to apply the resolution strategy. During this process of translating embedded representations into a flat structure such as clause form, the system loses the information about the rule nesting.

For instance, in a resolution-based system, `Rule1` is translated into clause form by a sequence of transformations :

$\neg$ `transitive(R)` $\vee$ $\neg$ `holds(R,x,y)` $\vee$ $\neg$ `holds(R,y,z)` $\vee$ `holds(R,x,z)`[2]

There is no difference among `Rule1`, $\texttt{Rule1}_a$, and $\texttt{Rule1}_b$ in this system since all three rules are uniformly transformed to the same clause form. Some specific rules might be migrated from this kind of system, but it has no ability to recognize which one is useful in a particular reasoning. This observation leads to our claim that any system which intends to realize the concept of migration is supposed to have a method of utilizing the characteristic of embedded representations with quantifiers.

3.7.3 Shadowing General Knowledge by Its Instances

This section proposes a method of recognizing general and specific knowledge from a collection of knowledge at various levels of generality. A scheme of shadowing is suggested to select the most specific knowledge whenever several candidates are waiting for rule activation. This is important to accomplish our goal of performance enhancement, and it also makes it possible to apply only domain dependent rules as far as we can in the expert system applications.

A motivation for the idea of shadowing is illustrated by reconsidering the transitive relation example. In Section 3.7.2, a natural deduction derivation is made to infer `supports(a,c)` from a given knowledge base. After the derivation, the system experiences a knowledge augmentation by asserting `supports(a,c)` and `Rule2` into the knowledge base. The expanded knowledge base is shown in Fig 3.7.1.

Now we want to derive another implicit fact `supports(b,d)` from this knowledge

[2] `holds` predicate is introduced to be consistent with a first-order logic

base. A natural deduction for this problem is expected to be divided into two branches, where one branch of the derivation starts from **Rule1** just like the previous derivation of **supports(a,c)**, and the other branch considers **Rule2** first. A sequence of the deduction in the first branch is described as :

Prop5 : {**transitive(supports)** $\Rightarrow$
$\forall$**x,y,z** {**supports(x,y)** & **supports(y,z)** $\Rightarrow$ **supports(x,z)**}}
from Rule1 by Universal Instantiation
with a binding {supports/R}
$\Downarrow$
Prop6 : $\forall$**x,y,z** {**supports(x,y)** & **supports(y,z)** $\Rightarrow$ **supports(x,z)**}
from Prop5 and Fact1 by Modus Ponens
$\Downarrow$
Prop7 : {**supports(b,c)** & **supports(c,d)** $\Rightarrow$ **supports(b,d)**}
from Prop6 by Universal Instantiation
with a binding {b/x, c/y, d/z}
$\Downarrow$
Prop8 : {**supports(b,d)**}
from Prop7, Fact3, and Fact4 by AND-introduction
and Modus Ponens

The second deduction branch is described as :

Prop9 : {**supports(b,c)** & **supports(c,d)** $\Rightarrow$ **supports(b,d)**}
from Rule2 by Universal Instantiation
with a binding {b/x, c/y, d/z}
$\Downarrow$
Prop10 : {**supports(b,d)**}
from Prop9, Fact3, and Fact4 by AND-introduction
and Modus Ponens

Note that these two branches form a OR-branch such that **supports(b,d)** can be derived by either one of two branches.

Several important points are worth mentioning from this observation. First of all, there are some duplicate reasoning steps in the first branch using **Rule1**, compared with the derivation of **supports(a,c)**. The similarity in the reasoning steps between these two derivations is expected since two reasonings share the same constant **supports**, which means they are in the same specific domain of **supports**. Another notable phenomenon is in the second branch. The reasoning steps in this branch are reduced to 2 for the derivation of **supports(b,d)**, compared with 4 steps in the previous inference of **supports(a,c)** and also in the first branch of **supports(b,d)**. Nothing goes better if we should be able to activate only this second branch with cutting off the first branch. This is the main objective of the shadowing scheme.

The issue now is how to systematically relate more general knowledge to its instances.

In order to take advantage of the previously acquired information, we need a way of memorizing instances with respect to the corresponding general knowledge. At this point, a method of saving the instances looks important. What we suggest to maintain is a list of instance information for each rule. Each instance element in the list has at least two kinds of information : the identification or the name of the instance, and a binding information explaining how the instance is related to the rule.

More formally, an instance list for a rule G has the form of

$$((S_1, \sigma_1), (S_2, \sigma_2), \cdots, (S_n, \sigma_n)),$$

where n is the number of known instances of G, S_i is the name of the i^{th} instance of G, and σ_i represents a binding which unifies G and S_i. In fact, S_i can be either a name, or a pointer to the actual structure of the i^{th} instance depending on particular implementations. All lists are initially empty, and they are dynamically updated as the inference goes on. In the transitive relation example, `Rule1` will be attached with an instance list after deriving `Prop2` such as `((Rule2, {supports/R}))`.

Once we defined the method of storing instances, the next step is to formulate a way of how to use them. Suppose a rule P has a list of known instances $((S_1, \sigma_1), (S_2, \sigma_2), \cdots, (S_n, \sigma_n))$ as defined above. Each binding in a known instance list σ_i contains only free variable substitutions. Also assume fv-list = $\{ fv_1, fv_2, \cdots, fv_m \}$ is a list of free variables of P. If, at some stage in a derivation, a proposition is deduced from P by universal instantiation with a binding ϕ, we check the information in ϕ with each σ_i $(1 \leq i \leq n)$ before any further action is made. We can stop this branch of derivation if the condition for shadowing is satisfied :

The rule P will be shadowed from the inference in the case that for any one of $i(1 \leq i \leq n)$, each substituted value for a free variable in σ_i is the same as the value for the same variable in ϕ.

Determining whether a variable in ϕ is free is done by checking the list membership for fv-list. For instance, consider the first deduction branch of `supports(b,d)`. When `Prop5` is being made from `Rule1`, we have a situation in which

ϕ = {supports/R}
σ = {supports/R}
fv-list = {R}.

Since R is the only free variable of `Rule1`, and the bound values for R in ϕ and σ are the same, the shadowing condition is satisfied. At this point, we have enough evidence that there is a more specific rule solely capable of deriving the given fact more efficiently. Therefore, the first branch is blocked here and will not be proceeded any further.

The evaluation of the shadowing may be made in two ways. Firstly, we can anticipate the shadowing makes the inference of `supports(b,d)` faster than in a non-shadowing normal inference because it is clear that there is some reductions of reasoning steps with the help of shadowing. Secondly, we hope that the improvement goes further so that the

inference of `supports(b,d)` is even faster than the first inference of `supports(a,c)` if the shadowing is adopted. Consequently, the system is now equipped with some intelligence which automatically prunes some inference branches using the previous reasoning experience.

3.7.4 An Implementation : SNePS

SNePS is a knowledge representation/ reasoning system using an intentional semantic network formalism [Shapiro, 1979]. SNePS is equipped with the ability to represent nested rules in any levels of depth. The inference package of SNePS (SNIP) provides an object-orient style of reasoning with assigning a process to each network node and maintaining several registers to keep information necessary for the message passing between processes. These registers are useful to save the instance information for shadowing.

Some peculiar features embodied in SNIP [Hull, 1986] are described in detail.

- SNIP treats inference as an activation of the network itself, rather than a compilation of the network into a distinct active connection graph of processes. (The latter method was adopted in old version of SNIP [Shapiro, 1977])
- There is a smaller set of processes and the types of processes are limited to the types of nodes found in the network. Current version of SNIP has 3 types of processes, that is `a proposition node process`, `a rule node process`, and `a user process`.
- Node processes are directly attached to the network nodes and the communications are made through `channels` which are incoming and outgoing paths between processes.

There are two types of messages which will be sent between node processes, **reports** and **requests.** The **reports** message contains substitutions which represent instances which are known to be true. The **requests** message contains desired substitutions, and the necessary information to set up the channels through which reports of these instances can be sent.

Each node process has a set of registers which actually set up the channel for message passing. Processes send and receive messages, and perform inferences based only on these messages and the register information. Some of the registers are described here.

- **known-instances** - the collection of instances of this node which are known to be true (both positive and negative)
- **reports** - the collection of reports received from other node processes
- **requests** - the collection of requests received from other node processes
- **incoming-channels** - the set of channels which will be feeding instance reports to this node
- **outgoing-channels** - the set of channels to which this node is to report instances that are discovered

A view of a message passing between two processes is shown in Figure 3.7.2.

Figure 3.7.2: A message passing between SNIP processes

To illustrate how SNIP realize the mechanism of the migration and the shadowing, we visit the transitive rule example again. Figure 3.7.3 (a) shows SNePSUL (SNePS User Language) format to represent the knowledge used in the example, and SNePS internal representation for these rules are described in Figure 3.7.3 (b). Here M represents a molecular node, and ! symbol indicates that the node is asserted at the top level. P and V denote a pattern node and a variable node, respectively [Shapiro, 1979].

Actual reasoning for `supports(a,c)` is done by **deduce** command as shown in Figure 3.7.3 (a), which builds an unasserted node `M6` and initiates a backward chaining to derive M6 from the given knowledge base.

SNIP assigns a process to each SNePS node when it is involved during the inference. The inference is performed solely by message passing between processes via channels. Channels between two processes are created if their corresponding nodes are pattern matched or, in case of rules, one node is unifiable with the consequent of the other node. Those channels made by rules are used for implementing rule chaining. The deduction of `supports(a,c)` in SNIP proceeds with the following steps.

Initially a user process is created and invokes process `M6`[3]. `M6` sends a request to `P4` by setting the **requests** register of `P4` to a substitution {`supports/V1, a/V2, c/V4`}. Since `P4` is the consequent of `P5`, `P4` sends the request to `P5` with {`supports/V1, a/V2, c/V4`}. `P5` is a rule node, but no known instances exist. So `P5` sends the request to its dominating rule node `M1!` with {`supports/V1`}, since `V1` is the only free variable in `P5`. `M1!` is an asserted rule node. So `M1!` sends the request to its antecedent `P1` with {`supports/V1`}. `P1` is matched with asserted proposition node `M5!`. `P1` sends a report back to `M1!` by setting the **reports** register of `M1!` to `M5!`, which then is sent back to `P5`. Now the free variable of `P5` is bound to **supports**. A migration takes place at this point. A new rule `M7!`, which is an instance of `P5`, is now asserted and the **known-instances** register of `P5` is set to `M7!`. The SNePS representation of the newly instantiated rule `M7!` is shown below.

[3]Processes are named after their corresponding SNePS node names.

```
(assert forall $r
        ant (build member *r class transitive)
        cq (build forall ($x $y $z)
                  &ant ((build agent *x act *r object *y)
                        (build agent *y act *r object *z))
                  cq (build agent *x act *r object *z)))

(assert agent a act supports object b)
(assert agent b act supports object c)
(assert agent c act supports object d)
(assert member supports class transitive)
(deduce agent a act supports object c)
```

(a)

```
(M1! (FORALL V1)
     (ANT (P1 (MEMBER V1) (CLASS TRANSITIVE)))
     (CQ (P5 (FORALL V2 V3 V4)
             (&ANT (P2 (AGENT V2) (ACT V1) (OBJECT V3))
                   (P3 (AGENT V3) (ACT V1) (OBJECT V4)))
             (CQ   (P4 (AGENT V2) (ACT V1) (OBJECT V4))))))

(M2! (AGENT A) (ACT SUPPORTS) (OBJECT B))
(M3! (AGENT B) (ACT SUPPORTS) (OBJECT C))
(M4! (AGENT C) (ACT SUPPORTS) (OBJECT D))
(M5! (MEMBER SUPPORTS) (CLASS TRANSITIVE))
(M6 (AGENT A) (ACT SUPPORTS) (OBJECT C))
```

(b)

Figure 3.7.3: SNePS representations for the transitive relation example

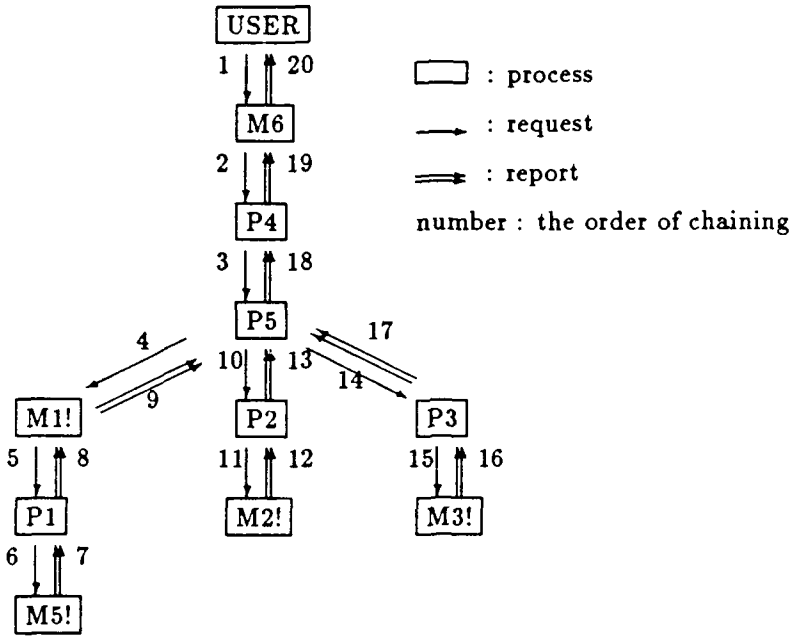


Figure 3.7.4: Process activation graph for **supports(a,c)**

```
(M7! (FORALL V2 V3 V4)
     (&ANT (P6 (AGENT V2) (ACT SUPPORTS) (OBJECT V3))
           (P7 (AGENT V3) (ACT SUPPORTS) (OBJECT V4)))
     (CQ (P8 (AGENT V2) (ACT SUPPORTS) (OBJECT V4))))
```

The exact content of the **known-instances** register of P5 will be :

```
*KNOWN-INSTANCES* = ((((P5 . M7!) (V1 . SUPPORTS)) . SNIP::POS))
```

This says M7! is a positive instance of P5 with the binding of {supports/V1}. Notice that the **known-instances** register has not only the name of the instance, but also the binding information for free variables, which is necessary to verify the appropriate instances. After migration, the inference goes on. P5 now sends a request to its antecedents, P2 and P3. Since P2 and P3 are matched with M2! and M3!, respectively, reports are sent from P2 and P3 to P5. The report from P5 is sent back to P4, then to M6, and then to the user process. Finally M6 is asserted as M6!. This process activation with messages passing through channels is drawn in Figure 3.7.4.

Now suppose we want to infer supports(b,d). SNePS builds M8 for this node and starts a deduction.

```
(M8 (AGENT B) (ACT SUPPORTS) (OBJECT D))
```

The inference procedure becomes more complicated because the knowledge base now has a migrated rule M7! as well as M1!.

Initially a user process is created and invokes process M8. M8 is matched with both P4 and P8. So now the inference goes with two branches. While a request is sent from M8 to

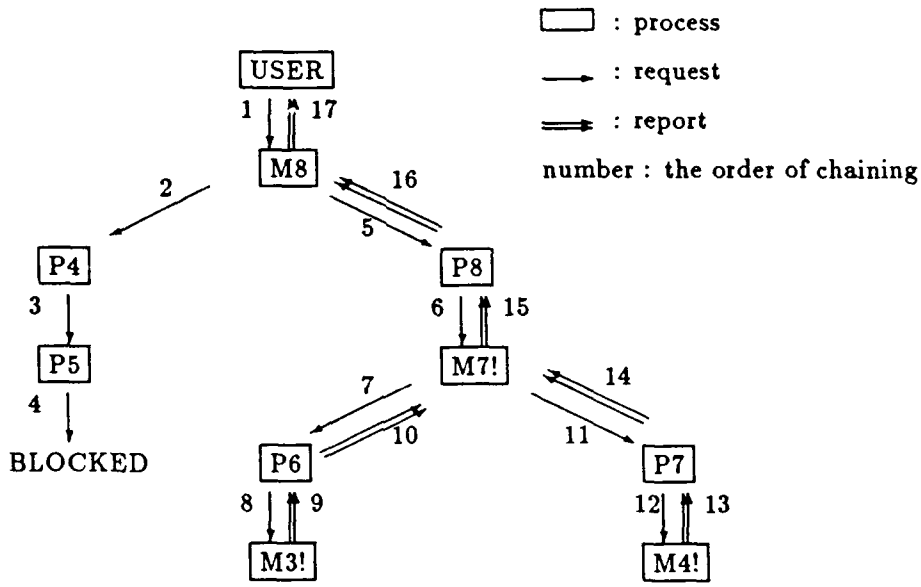


Figure 3.7.5: Process activation graph for **supports(b,d)**

P4 with a substitution {supports/V1, b/V2, d/V4}, M8 sends another request to P8 with a different substitution {b/V2, d/V4}. Note that there is no substitution for V1 because P8 has no such variable. Since P4 is the consequent of P5, P4 sends the request to process P5 with {supports/V1, b/V2, d/V4}. P8 is also the consequent of M7!, so P8 sends the request to M7! with {b/V2, d/V4}. These steps of requests sending can proceed in parallel.

When process P4 is sending a request to P5 with the substitution of {supports/V1, b/V2, d/V4}, the **requests** register of P5 is set to :

```
*REQUESTS* = ((((P4 . M8) (V1 . SUPPORTS) (V4 . D) (V2 . B))
               NIL  P4  OPEN))
```

This structure tells that a request is sent from P4 via an open channel, and the requested substitution is {supports/V1, b/V2, d/V4}. Now we check the shadowing condition mentioned in Section 3.7.3. Since the process P5 has M7! as an instance in the **known-instances** register, the next step is to verify that M7! is a useful instance in this particular inference. The filtering process compares the binding information for free variables in both registers. Eventually M7! is accepted as a proper instance because the substitution for free variable V1 of P5 in the **requests** register is identical to that in the **known-instances** register. Finally the activation of P5 is blocked and the process M7! will be responsible for the remaining inference. Process activation graph for this part is drawn in Figure 3.7.5.

We ran this example by SNIP on TI Explorer, and Table 3.7.1 shows the time comparisons for the execution of these rules. We compare the time between supports(a,c) and supports(b,d), and also between the case that the shadowing is implemented and the case without shadowing. The execution of supports(b,d) is done after the more specific rule is generated by supports(a,c).

unit : seconds

	without shadowing	with shadowing
`supports(a,c)`	6.85	5.92
`supports(b,d)`	10.37	3.85

Table 3.7.1: Execution time comparisons for the transitive relation rule

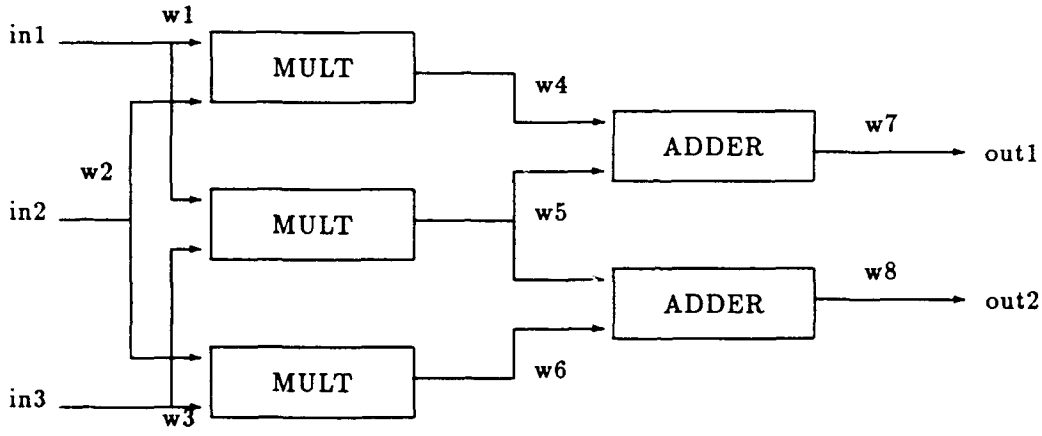


Figure 3.7.6: A logical abstraction for M3A2

In this table, we will see the reduced time for `supports(b,d)` from 10.37 to 3.85 by shadowing. Furthermore, we can also notice from the column named `with shadowing` that the inference of `supports(b,d)` is even faster between `supports(a,c)`. This result tells the real performance enhancement obtained by applying the scheme of shadowing.

3.7.5 An Application

Some diagnostic expert systems use a model of devices which is structural or functional [Davis, 1984, Taie, 1987]. This approach has been used to find a faulty component in a digital combinational circuit like M3A2. M3A2 is a simple circuit which has 3 multipliers and 2 adders as shown in Figure 3.7.6.

The values of outputs are determined by inputs as :

```
out1 = in1 * in2 + in1 * in3
out2 = in1 * in3 + in2 * in3
```

If the calculated values of outputs from given inputs are different from the measured values, a violation is detected and the diagnosis starts to locate the faulty components. The component could be a device or a wire, so we need diagnostic rules for such components.

An example we will show is a wire faulty detection for M3A2. There are several types of wires used in this circuit analysis. For instance, WIRE3 denotes a type of wires connected

```
(M2! (FORALL V1)
     (ANT (P1 (TYPE V1) (TYPE-CLS  WIRE)))
     (CQ (P11  (FORALL V2 V3 V4)
               (&ANT (P2 (OBJECT V2) (TYPE V1))
                     (P3 (BI-PORT1 V2) (OBJECT V3))
                     (P4 (BI-PORT2 V2) (OBJECT V4)))
               (CQ (P10  (FORALL V5 V6)
                         (&ANT (P6 (OBJECT V3)
                                   (ATTR (P5  (ATRB V5)
                                              (ATRB-CLS  M-VALUE)
                                              (MODALITY LOGICAL))))
                               (P8 (OBJECT V4)
                                   (ATTR (P7  (ATRB V6)
                                              (ATRB-CLS  M-VALUE)
                                              (MODALITY LOGICAL)))))
                         (CQ (P9 (OBJECT V2) (TYPE V1)
                                 (ATTR (M1  (ATRB FAULTY)
                                            (ATRB-CLS  STATE)
                                            (MODALITY LOGICAL)))))))))))
```

Figure 3.7.7: SNePS representation for a wire-faulty detection rule

to three different components, and WIRE2 denotes a different type of wires connected to two different components. In Figure 3.7.6, w4, w6, w7, and w8 fall under the category of WIRE2, and w1, w2, w3, and w5 have WIRE3 property. We can build a diagnose rule for detecting wire faults which generally applies to different types of wires.

$$\forall T \{ \text{Wire-type}(T) \Rightarrow \forall O,P1,P2 \{ T(O) \,\&\, \text{Bi-port}(O,P1,P2) \,\&\, \text{value}(P1) \neq \text{value}(P2) \Rightarrow \text{faulty}(O) \}\}$$

A SNePS representation for this rule is shown in Figure 3.7.7.

Suppose the first diagnose is for w7 of WIRE2. After migration, a specific rule is generated for wires of type WIRE2 by replacing the free variable T of the general rule by WIRE2.

$$\forall O,P1,P2 \{ \text{Wire2}(O) \,\&\, \text{Bi-port}(O,P1,P2) \,\&\, \text{value}(P1) \neq \text{value}(P2) \Rightarrow \text{faulty}(O) \}$$

A SNePS representation for this migrated rule is shown in Figure 3.7.8.

This rule will shadow the original rule when another reasoning is performed for a wire of the same type w8. Table 3.7.2 show the improvement of performance by comparing two executions with or without shadowing.

Shadowing can be very effective especially for the applications which perform diagnosis about similar components many times. So far we illustrate the potential applicability of the migration and the shadowing scheme to the real domain of applications. We would like

```
(M10! (FORALL V2 V3 V4)
      (&ANT (P2 (OBJECT V2) (TYPE WIRE2))
            (P3 (BI-PORT1 V2) (OBJECT V3))
            (P4 (BI-PORT2 V2) (OBJECT V4)))
      (CQ (P10  (FORALL V5 V6)
                (&ANT (P6 (OBJECT V3)
                          (ATTR (P5  (ATRB V5)
                                     (ATRB-CLS  M-VALUE)
                                     (MODALITY LOGICAL))))
                      (P8 (OBJECT V4)
                          (ATTR (P7  (ATRB V6)
                                     (ATRB-CLS  M-VALUE)
                                     (MODALITY LOGICAL)))))
                 (CQ (P9 (OBJECT V2) (TYPE WIRE2)
                         (ATTR (M1  (ATRB FAULTY)
                                    (ATRB-CLS  STATE)
                                    (MODALITY LOGICAL))))))))
```

Figure 3.7.8: SNePS representation for a migrated wire-faulty rule

unit : seconds

	without shadowing	with shadowing
faulty(w7)	39.18	38.93
faulty(w8)	43.22	27.10

Table 3.7.2: Execution time comparisons for wire faulty detection rule

to explain in the next section the details about the implementation.

3.7.6 Conclusion

An automatic scheme is suggested for migrating specific knowledge and for shadowing deep knowledge by its instances in an expert reasoning system. The motivation of this work is to make the inference faster in a multi-level knowledge system in which various kinds of knowledge are present. Migration and shadowing schemes enable the system to accomplish the performance improvement as well as the system generalization. Most specific knowledge is preferred to be selected among candidates at each stage of the inference. Experimental results have shown that the inference speed is significantly improved with shadowing method. Applicability to real complex domain should be further tested.

3.8 CONCLUSION

We have developed a prototype system called Versatile Maintenance Expert System (VMES) to diagnose a variety of common electrical faults in electronic circuits. This system easily adapts to new devices. VMES consists of an integrated knowledge base and a device independent inference engine. The inference engine is implemented in Semantic Network Processing System (SNePS). The current version of VMES demonstrates its strength in diagnosis by incorporating features such as Model-Based Reasoning, and communication capabilities such as Natural Language and Graphics. It is also capable of diagnosing sequential components and systems of the complexity of a microcomputer.

3.8.1 Accomplishments

VMES, when it was first conceived in 1984, was designed as a rule-based system to advise a maintenance technician on testing. However, it has been felt that versatility is extremely important in an electronic circuit domain due to the fast rate at which new products are introduced and their relatively short market life. VMES was built to be versatile across a broad range of target devices in the circuit domain, across most of the possible faults, across different maintenance levels and across a variety of user interfaces. In order to achieve such versatilities and to avoid the difficulties of empirical rule-based diagnosis, VMES adopted a device model-based approach for diagnosis. VMES uses both structure and functional descriptions of devices by modeling a device in the circuit domain as a hierarchically arranged set of subparts from both logical and physical perspectives.

A major step in model-based fault diagnosis has been the generation of candidate submodules which might be responsible for the observed symptom of malfunction. After the candidates are determined, each submodule can then be examined in turn. It is useful to be able to choose the most likely candidate to focus on first so that the faulty parts can be located sooner. We have developed a systematic method for candidate ordering that takes into account the structure of the device and the discrepancy in outputs between the observed and expected values. More dynamic methods called candidate reordering and elimination have been developed to overcome the limitations of initial candidate ordering. The new method modifies the candidate list as new information becomes available.

Sequential circuit diagnosis is another major aspect of VMES. In order to incorporate sequential circuit diagnosis into VMES, the following steps have been taken: (1) change in device knowledge representation; (2) change in control structure and inclusion of assumption relaxation; and (3) candidate generation based on electrical behavior. The change in representation was essential for better organization of the device knowledge and the incorporation of sequential components. Diagnosis is based on logical hierarchy and the relation between the logical and physical hierarchies has been deemphasized. This has enabled arbitrary number of logical levels and has allowed arbitrary grain of focus for diagnosis.

A significant aspect of the later part of VMES research is the consideration of a complex

commercial system as a test bed. Upon recommendation by RADC, a Heathkit Printer Buffer Board was selected. This test device, assembled locally, consists of an eight-bit microprocessor, two sets of serial and parallel ports, memory and latches. This circuit has been analyzed and represented at various levels of abstraction. At the topmost level, several rules concerning the failure of the device have been identified and incorporated in the system. At the intermediate level, certain failure modes have been identified and functional knowledge of some of the modules is obtained. A device of this kind helped spur new ideas, extensions and refinements to the diagnosis theory.

During the course of this project, there has been continuing development of SNePS. Migration of deep knowledge to shallow knowledge to enhance the speed of inference has been researched during the later part of this project. Some of the possible extensions to our research is discussed next.

3.8.2 Possible Extensions

The current VMES research has fulfilled our original objectives of efficient device knowledge representation and versatile diagnosis. It has successfully diagnosed circuits of the complexity of a four-bit magnitude comparator, a signal converter circuit (PCM4), and a four-bit sequential multiplier. It has the capability of diagnosing circuits of the complexity of a printer buffer board. However, VMES has some limitation in actual diagnosis of circuits of arbitrary complexity. This is due to the fact that the acquisition and representation of device knowledge at arbitrary grain of abstraction is quite complex. Due to the nonavailability of the design and simulation details of the printer buffer board, a complete diagnosis of this circuit was not possible in the given time frame. A good understanding of the design of a given circuit is required to perform diagnosis of complex circuits.

There are a number of ways by which the capabilities of VMES could be enhanced. We discuss two categories of enhancements: (1) enhancements of the diagnosis theories developed in this project, and (2) development of new theories to advance the concept of diagnosis.

We first discuss category (1). The candidate generation theory developed in this research is based on the assumption that only a single symptom is available. Our theory can be refined to include multiple symptoms, test generation and circuits with feedback loops. The reasoning methodology in model-based diagnosis can be improved by incorporating heuristic knowledge gained by technician's experience. The unification of procedural and declarative knowledge in diagnosis is also a problem worthy of consideration. An example for such a requirement is the concept of board swapping which requires both algorithmic and heuristic knowledge.

Most commercial systems are sequential in nature. Therefore, diagnosing systems with internal states is an important problem. Sequential circuits should be represented in temporal, functional and structural levels. More research is required to represent the functional knowledge of sequential circuits. Understanding of systems from its design perspective and utilizing that knowledge for the representation and diagnosis will be a new direction in

model-based diagnosis.

Enhancements in category (2) are summarized below: Most of the candidate generation techniques in the literature use a breadth-first or depth-first search to reduce the suspect set. More efficient algorithms such as the divide and conquer techniques can be employed to reduce the runtime diagnosis time. This is a new theory and needs investigation.

In today's systems *diagnosability* is often built into the systems. However, due to the tradeoff between the diagnosability and hardware costs associated with it, systems have very limited diagnosability. Model-based diagnosis in such systems can benefit from the *diagnosable* and *testable* design details. Another future direction in this area is the utilization of *error logging information.* Error logging details usually contain some information about the internal states of the system which can be used in diagnosis.

The current version of VMES does not consider the diagnosis of analog circuits. Analog circuit diagnosis will be different from digital circuit diagnosis since there is no clear-cut hierarchy in analog circuits. However, due to the mix of analog and digital circuitry in most of the systems, diagnosis of analog circuits will be a significant advance. Diagnosis of intermittent faults is another area that needs further investigation.

The control structure of VMES can be generalized to include various schemes of diagnosis. For instance, a *retry* method, or *direct isolation* or *intersection isolation* can be applied in a sequence to diagnose a circuit. This involves the development of direct and intersection isolation techniques and its implementation.

VMES project has also seen several enhancements in SNePS, the system used to implement VMES. Concepts such as migration of deep knowledge to shallow knowledge and shadowing rules will be pursued further to speed up the inference mechanism of VMES.

One of the major goals of our research was to design a system that is versatile and capable of diagnosing circuits of the complexity of a microcomputer and beyond. Although, VMES in its current form has such a capability, more research is required to make VMES applicable to practical circuits of arbitrary complexity.

3.9 REFERENCES

[Barbacci and Uehara, 1985] M. R. Barbacci and T. Uehara. Computer hardware description languages: The bridge between software and hardware. *Computer*, pages 6–9, February 1985.

[Bibel, 1986] Wolfgang Bibel. Methods of automated reasoning. In W. Bibel and Ph. Jorrand, editors, *Fundamentals of Artificial Intelligence: An Advanced Course*, pages 171–217. Springer-Verlag, 1986.

[Buchanan and Shortliffe, 1984] B. G. Buchanan and E. H. Shortliffe, editors. *Rule-based Expert Systems.* Addison-Wesley, Reading, MA, 1984.

[Campbell and Shapiro, 1986] S. S. Campbell and S. C. Shapiro. Using belief revision to detect faults in circuits. SNeRG Tech. Note 15, Department of Computer Science, SUNY at Buffalo, Buffalo, NY 14260, 1986.

[Chandrasekaran and Mittal, 1983] B. Chandrasekaran and Sanjay Mittal. Deep versus compiled knowledge approaches to diagnostic problem solving. *International Journal of Man-Machine Studies*, 19:425–436, 1983.

[Chen and Srihari, 1989] J. Chen and S. N. Srihari. Candidate ordering and elimination in model-based fault diagnosis. In *Proceedings of IJCAI.* Morgan Kaufmann, 1989.

[Chu, 1974] Y. Chu. Why do we need hardware description languages? *Computer*, pages 18–27, December 1974.

[Clancey, 1983] W. J. Clancey. The epistemology of a rule-based expert system: A framework for explanation. *Artificial Intelligence*, 20:215–251, 1983.

[Davis and Shrobe, 1983] R. Davis and H. Shrobe. Representing structure and behavior of digital hardware. *Computer*, 16(10):75–82, October 1983.

[Davis, 1983] R. Davis. Diagnosis via causal reasoning: Paths of interaction and the locality principle. In *Proceedings of AAAI*, pages 88–94, 1983.

[Davis, 1984] R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24(3):347–410, 1984.

[Fahlman, 1979] S. E. Fahlman. *NETL: A System for Representing and Using Real-world Knowledge.* The MIT Press, 1979.

[Feigenbaum *et al.*, 1971] E. A. Feigenbaum, B. G. Buchanan, and J. Lederberg. On generality and problem solving: A case study using the DENDRAL program. In B. Meltzer and D. Michie, editors, *Machine Intelligence 6*, pages 165–190. Elsevier, New York, 1971.

[Feigenbaum, 1979] E. A. Feigenbaum. Themes and case studies of knowledge engineering. In D. Michie, editor, *Expert Systems in the Micro Electronic Age*, pages 3–25. Edinburgh University Press, 1979.

[Geller, 1988] J. Geller. *A Knowledge Representation Theory for Natural Language Graphics.* PhD thesis, Department of Computer Science, SUNY at Buffalo, 1988. Technical Report 88-15.

[Genesereth, 1982] M. R. Genesereth. Diagnosis using hierarchical design models. In *Proceedings of AAAI*, pages 278–283, 1982.

[Genesereth, 1984] M. R. Genesereth. The use of design description in automated diagnosis. *Artificial Intelligence*, 24(3):411–436, 1984.

[German and Lieberherr, 1985] S. M. German and K. J. Lieberherr. Zeus: A language for expressing algorithm in hardware. *Computer*, pages 55–66, February 1985.

[Hamscher and Davis, 1984] W. Hamscher and R. Davis. Diagnosing circuits with state: An inherently underconstrained problem. In *Proceedings of AAAI*, pages 142–147. Morgan Kaufmann, 1984.

[Hart, 1982] Peter E. Hart. Directions for AI in the eighties. *SIGART Newsletter*, 79:11–16, January 1982.

[Hayes-Roth *et al.*, 1983] F. Hayes-Roth, D. A. Waterman, and D. B. Lenant, editors. *Building Expert System.* Addison-Wesley, 1983.

[Hull, 1986] Richard G. Hull. A new design for SNIP the SNePS inference package. SNeRG Technical Note 14, Dept. of Computer Science, State University of New York at Buffalo, 1986.

[McDermott and Forgy, 1978] J. M. McDermott and C. Forgy. Production system conflict resolution strategies. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern Directed Inference Systems.* Academic Press, New York, 1978.

[Michie, 1980] D. Michie. Expert systems. *The Computer Journal*, 23(4):369–376, 1980.

[Quinlan, 1982] J. R. Quinlan. Fundamentals of the knowledge engineering problem. In D. Michie, editor, *Introductory Readings in Expert Systems*, pages 33–46. Gordon and Breach, New York, 1982.

[Rasmussen and Jensen, 1974] J. Rasmussen and A. Jensen. Mental procedures in real-life tasks: A case study of electronic trouble shooting. *Ergonomics*, 17:293–307, 1974.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[Robinson, 1965] J. A. Robinson. A machine oriented logic based on the resolution principle. *Journal of ACM*, 12:23–41, 1965.

[Rosch, 1978] E. Rosch. Principles of categorization. In E. Rosch and B. Lloyd, editors, *Cognition and Categorization*, pages 27–48. 1978.

[Sauers, 1988] Ron Sauers. Controlling expert systems. In L. Bolc and M. J. Coombs, editors, *Expert System Applications*, pages 79–197. Springer-Verlag, 1988.

[Sembugamoorthy and Chandrasekaran, 1986] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem solving systems. Technical Research Report 86-VS-FUNCT, Dept. of Computer and Information Science, The Ohio State University, 1986.

[Shapiro, 1977] Stuart C. Shapiro. Compiling deduction rules from a semantic network into a set of processes. In *Abstracts of Workshop on Automatic Deduction*, MIT, 1977.

[Shapiro, 1979] Stuart C. Shapiro. SNePS: Semantic network processing system. In N.V. Findler, editor, *Associative Networks: Representation and Use of Knowledge by Computers*, pages 179–203. Academic Press, New York, 1979.

[Shirley and Davis, 1983] M. H. Shirley and R. Davis. Generating distinguishing tests based on hierarchical models and symptom information. In *Proceedings of IEEE International Conference on Computer Design*, pages 455–458, 1983.

[Shortliffe, 1976] E. Shortliffe. *Computer Based Medical Consultations: MYCIN.* Elsevier, New York, 1976.

[Siewiorek, 1974a] D. Siewiorek. Introducing ISP. *Computer*, pages 39–41, December 1974.

[Siewiorek, 1974b] D. Siewiorek. Introducing PMS. *Computer*, pages 42–44, December 1974.

[Taie and Srihari, 1987] M. R. Taie and S. N. Srihari. Modeling connections for circuit diagnosis. In *Proceedings of the Third Conference on Artificial Intelligence Applications*, pages 81–86, Orlando, FL, February 1987. IEEE Computer Society Press.

[Taie, 1987] Mingruey R. Taie. Representation of device knowledge for versatile fault diagnosis. Technical Report 87-07, Department of Computer Science, State University of New York at Buffalo, 1987.

[Waterman, 1979] D. A. Waterman. User-oriented systems for capturing expertise: A rule-based approach. In D. Michie, editor, *Expert Systems in the Micro Electronic Age*, pages 26–34. Edinburgh University Press, 1979.

[Weiss *et al.*, 1978] S. M. Weiss, C. A. Kulikowski, S. Amarel, and A. Safir. A model-based method for computer-aided medical decision making. *Artificial Intelligence*, 11:145–172, 1978.

[Williams, 1986] C. Williams. Expert systems, knowledge engineering, and AI tools: An overview. *IEEE Expert*, pages 66–70, Winter 1986.

[Xiang and Srihari, 1985] Z. Xiang and S. N. Srihari. Spatial structure and function representation in diagnostic expert systems. In *Proceedings of the Fifth International Workshop on Expert Systems and Their Applications*, pages 191–206, Avignon, France, May 1985.

[Xiang and Srihari, 1986] Z. Xiang and S. N. Srihari. A strategy for diagnosis based on empirical and model knowledge. In *Proceedings of the Sixth International Workshop on Expert Systems and Their Applications*, pages 835–848, Avignon. France, April 1986.

[Yoon and Hammer, 1988] W. C. Yoon and J. M. Hammer. Deep-reasoning fault diagnosis: An aid and a model. *IEEE Trans. on Systems, Man, and Cybernetics*, 18(4):659–676, July/August 1988.

3.10 VMES PUBLICATIONS (1985–1989)

Shapiro, S. C.and Srihari, S. N. and Taie, M. R. and Geller, J. "Development of an Intelligent Maintenance Assistant", *ACM SIGART Newsletter* No. 92, P 48-49, April 1985.

Shapiro, S. C. and Srihari, S. N. and Geller, J. and Taie, M. R. "A Fault Diagnosis System Based on an Integrated Knowledge Base", *IEEE Software* V 3, N 2, P 48-49, March 1986.

Shapiro, S. C. and Srihari, S. N. and Taie, M. R. and Geller, J. "VMES: A Network-Based Versatile Maintenance Expert System", *Proc. of 1st International Conference on Applications of AI to Engineering Problems* P 925-936, Springer-Verlag, New York, April 1986.

Taie, M. R. and Srihari, S. N. and Geller, J. and Shapiro, S. C. "Device Representation Using Instantiation Rules and Structural Templates", *Proc. of Canadian AI Conference - 86* P 124-128, Presses de I'Universite du Quebec, Montreal, Canada, May 1986.

Shapiro, S. C. and Geller, J. "Knowledge Based Interfaces", *AAAI-86 Workshop on Intelligence in Interfaces* Bob Neches, Tom Kaczmarek, P 31-36, August 14, 1986.

Taie, M. R. and Srihari, S. N. "Device Modeling for Fault Diagnosis", *Proc. of the 2nd Expert Systems in Government Symposium* P 144-150, IEEE Computer Society Press, Washington, D. C., October 1986.

Shapiro, S. C. "Symmetric Relations, Intensional Individuals, and Variable Binding", *Proceedings of the IEEE 74* P 1354-1363, October 1986.

Shapiro, S. C. and Geller, J. "Artificial Intelligence and Automated Design", *Proc. of the SUNY Buffalo Symposium on CAD: The Computability of Design* SUNY at Buffalo, NY, December 1986.

Taie, M. R. and Geller, J. and Srihari, S. N. and Shapiro, S. C. "Knowledge Based Modeling of Circuit Boards", *Proc. of 1987 Annual Reliability and Maintainability Symposium* P 422-427, IEEE, Philadelphia, PA, January 1987.

Taie, M. R. and Srihari, S. N. "Modeling Connections for Circuit Diagnosis", *Proc. of The 3rd IEEE Conference on AI Applications* P 81-86, IEEE Computer Society Press, Orlando, FL, February 1987.

Srihari, S. N. "Applications of Expert Systems in Engineering", *Engineering Progress of Western New York 6, 2* P 17-21, Faculty of Engineering and Applied Sciences, SUNY at Buffalo, Winter-Spring 1987.

Taie, M. R. "Representation of Device Knowledge for Versatile Fault Diagnosis", *Technical Report 87-07 (Ph.D. Dissertation)* Department of Computer Science, SUNY at Buffalo, Buffalo, NY, May 1987.

Geller, A. J. and Taie, M. R. and Shapiro, S. C. and Srihari, S. N. "Device Representation and Graphics Interfaces of VMES", *Knowledge Based Expert Systems for Engineering: Classification, Education and Control (a paper collection from the Second International Conference on Applications of AI in Engineering)* D. Sriram, R.A. Adey, P 15-28, Computational Mechanics Publications, Boston, MA, August 1987.

Geller, J. and Shapiro, S. C. "Graphical Deep Knowledge for Intelligent Machine Drafting", *Proc. of the 10th International Joint Conference on Artificial Intelligence* P 545-551, Morgan Kaufmann, Milan, Italy, August 1987.

Shapiro, S. C. and Geller, J. "Artificial Intelligence and Automated Design", *Computability of Design* Yehuda E. Kalay, P 173-187, John Wiley & Sons, New York, NY, 1987.

Geller, J. "A Knowledge Representation Theory for Natural Language Graphics", *Technical Report 88-15, (Ph.D. Dissertation)* Department of Computer Science, SUNY at Buffalo, Buffalo, NY, July 1988.

Srihari, S. N. "Applications of Expert Systems in Engineering", the first chapter of *Knowledge-Based System Diagnosis, Supervision and Control* S. G. Tzafestas, 1988.

Chen, J. S. and Srihari, S. N. "A Method for Ordering Candidate Submodules in Fault Diagnosis", *Technical Report TR-8736*, Northeast Artificial Intelligence Consortium, Syracuse, NY, 1988.

Srihari, S. N. and Xiang, Z. "Spatial Knowledge Representation", *Journal of Pattern Recognition and Artificial Intelligence*, Vol. 3, No. 1, March 1989.

Chen, J-S. and Srihari, S.N. "Candidate Ordering and Elimination in Model-based Fault Diagnosis", *Proceedings of the Eleventh Annual International Joint Conference on Artificial Intelligence* Detroit, Michigan, August 20–25, 1989.

3.11 TRIPS FUNDED BY RADC

Executive Committee Meeting and Principal Investigator Meeting, Rochester Institute of Technology, April 14, 1989: Srihari, Shapiro.

NAIC Committee Meeting, Syracuse, NY, June 28, 1989: Shapiro.

Annual RADC Meeting, Minnowbrook, NY, August 14–17, 1989: Srihari, Shapiro, Upadhyaya, Choi.

Eleventh International Joint Conference on Artificial Intelligence (IJCAI), Detroit, Michigan, August 20–25, 1989: Shapiro, Upadhyaya, Chen.

3.12 STUDENTS DIRECTLY FUNDED BY NAIC, 1985–1989

(Names in **bold** are supported in 1989 by RADC at SUNY Buffalo)

James Geller, Ph.D.	2/85 - 5/88	Research Assistant
Academia, USA	6/88 - 7/88	Research Associate
Permanent Resident (applied after graduation)		
Mingruey Taie, Ph.D.	2/85 - 5/87	Research Assistant
Industry, USA	5/87 - 8/87	Research Associate
Permanent Resident (applied after graduation)		
Joao P. Martins, Ph.D.	5/85 - 7/85	Research Associate
Academia, Portugal	Post Doc. work	Visiting Asst. Professor
Foreign		
Scott S. Campbell, MS	8/85 - 9/87	Research Assistant
American	9/87 - 8/89	Programmer/Analyst
Jiah-shing Chen, MS	7/87 - 8/89	Research Assistant
Foreign		
Amruth Kumar N.	1/88 - 8/89	Research Assistant
Foreign		
Joongmin Choi	8/88 - 8/89	Research Assistant
Foreign		
Deepak Kumar, MS	8/87 - 8/89	Research Assistant
Foreign		
Syed S. Ali, MS	8/87 - 8/89	Research Assistant
Canadian		
Juergen Haas	1/89 - 8/89	Research Assistant
Foreign		
Sudip Nag	8/88 - 7/89	Research Assistant
Permanent Resident		

Byung S. Yoo, MS Permanent Resident Industry, IBM	7/88 – 10/88	Research Assistant
David B. Satnik American Industry, Seattle, WA	7/87 – 5/88	Research Assistant
Richard W. Wyatt Permanent Resident Ph.D., Psychology Student, SUNY at Buffalo	8/87 – 1/88	Research Assistant
Keith E. Bettinger, MS American Student, SUNY at Buffalo	5/87 – 8/87	Research Assistant
Kwong Yiu Yim Foreign	1/86 – 5/86	Research Assistant
Jennifer M. Suchin, MS American Industry, Pittsburgh, PA	6/85 – 8/85	Project Aide (Graduate)

RADC Supported Ph.D.'s in Progress

Kumar, Deepak
Planning in SNePS

Chen, Jiah-shing
Model-based Diagnosis Using Multiple Approaches

Choi, Joongmin
An Intelligent Reasoning System by Knowledge Migration and Shadowing

Kumar N., Amruth
Issues in Diagnosis: Sequential and Combinational

3.13 DEPARTMENT STATISTICS: ARTIFICIAL INTELLIGENCE

YEAR	AI Ph.D.'s	Non-AI Ph.D.'s	AI Master's	Non-AI Master's
1989 (through June)	None	None	21	11
1988	5	None	23	12
1987	3	None	28	16
1986	1	2	24	7
1985	2	1	28	3
1984	None	2	27	9
1983	2	1		

3.14 PH.D. GRADUATES IN ARTIFICIAL INTELLIGENCE

(Names in **bold** were supported by RADC)

1988

James Geller	Assistant Professor	New Jersey Institute of Technology
Jonathan J. Hull	Research Assistant Professor	SUNY at Buffalo Buffalo, NY
Ganapathy Krishnan	Assistant Professor	Stetson University Deland, FL
Ching-Huei Wang	Research Analyst	Boeing Electronics Corp. Seattle, WA
Zhigang Xiang	Assistant Professor	Dept. of Computer Science Queen's College (CUNY) New York, NY

1987

Michael J. Almeida	Assistant Professor	Penn State University Dept. of Computer Science Whitmore Lab University Park, PA
George Sicherman		AT&T Bell Labs Middletown, NJ
Mingruey R. Taie		AT&T Bell Labs Middletown, NJ

1986

Ernesto Morgado	Assistant Professor	Dept. de Engenaria Mechanica Instituto Superior Tenico Lisbon, Portugal

1985

Radmilo M. Bozinovic		GO Corporation San Francisco, CA
Jeannette Neal		Calspan Corporation Cheektowaga, NY Research (CUBRC)

3.15 MASTER'S DEGREES FROM THE DEPARTMENT OF COMPUTER SCIENCE (1984–1989)

(Names in **bold** were supported by RADC)

Artificial Intelligence (1989, through June)

Ansley, William	Dobes, Zuzana	Majkowski, Bruce
Banerjee, Sarbani	Goldberg, Neal	**Nag, Sudip**
Chaluplsky, Hans	Gucwa, John R.	Po, Cherng-Fong
Chang, Han Yi	Jain, Naresh Kumar	Sin,Via Tong
Cohen, Edward	Lee, Chieng	Tan, Wan
Colucci, Paul	Lin, Yi Chang	Wu, Li Shin
Crovella, Mark E.	Lombardo, Karen	Yen, Shyh-Guang

CS Other than AI (1989, through June)

Arora, Rajendra	Grupka, Laurette	Sherman, Paul J.
Chang, Tien	Kingsbury, Linda A.	Williams, Francine
Delgado, Diane M.	Liu, Paul C-W	Wu, W-C Jevons
Fenrich, Richard	Menon, Rajeev	

Artificial Intelligence (1988)

Bansal, Surendra	Haller, Susan	Soh, Jung
Benz, David	Hardy, Michael J.	Strohmeier, Nancy
Bettinger, Keith	Hou, Chien-Long	Swerdloff, Lucien
Biernat, Catherine	Kornacki, Edmund	Vecellio, Gary
Chang, Adam Chih-Yen	Kuan, Chic Chau	Wan, Tzu-Horng Tom
Danko, Paul Jr.	**Kumar, Deepak**	Williams, Francine
Debbins, Catherine	Lew, Kurk	**Yoo, Byung**
Govindaraju, Venugopal	Lo, Ka-Chiu	

CS Other than AI (1988)

Arora, Rajenda	Duh, Ying Ying	Lam, William
Azar, Chawki	Hosangadi, Shrikant	Mantharam, Mythili
Bahl, Vikram	Lagona, Scott	Sarraf, Elias
Desirazu, Shyam	Lakshman, T.K.	Wahl, Norman

Artificial Intelligence (1987)

Binkerhoff, Linda
Chan, Chung Man
Campbell, Scott
Chang, Cheng-Ping
Chen, Yung-Yuan
Chun, Soon Ae
DeVinney, George
Dodson-Simmons, Onda
Ehrlich, Karen
Feuerstien, Steven
Gupta, Rakesh
Jain, Hwejdar
Kim, Joong-Won
Kuo, Chung-Kuo
Lang, Su-Jin
Lee, Hui-Chung
Li, Niacong
Li, Peter
Murty, Kurella
Schneck, Nelson
Schwartz, Margaret
Siracusa, Thomas
So, Hon-Man
Thomas, Timothy
Wang, Gretchen
Wroblewski, Susan
Wu, Teng Yien
Wu, Wei-Jye

CS Other than AI (1987)

Bo› ⌊awrence
Chang, Cheng-Ping
Cheng, Tony H-Y
Chow, Lawrence
Gaur, Yogesh
Girod, Allison
Gunning, Mark
Hiroi, Toshiyuki
Jang, Yong Ho
Lively, Richard
Mackey, Niloufer
Miller, Susan
Nimmagadda, Venkata
Rajan, Dayanand
Shende, Anil
Subrahmanyam, Pratap

Artificial Intelligence (1986)

Bross, Neal
Deutschlander, Kenneth
Hull, Richard
Jayanthi, Sarma
Kailar, Sudah
Krishnaswamy, Latha
Krishnaswamy, Vijaykumar
Lee, Gin-Wha
Lu, Wuhsiung
Ma, Pu-Kao
MacFadden, Douglas
McConnell, Jeffery
Murphey-Shelton, Anne
Murray, Deborah
Rastogi, Ajay
Sauciunac, Christine
Shin, Kwang Un
Shyong, Beth M-F
Swaminathan, Puducode
Ting, Hungtau
Wang, Fen-Cheng
Winkowski, Danie
Wood, Gabriel
Yang, Ching-Yun

CS Other than AI (1986)

Bharadhwaj, Rajeev
Kim, Dongsoo
Martin, Dennis
Ramshankar, J.V.
Rosenblum, Leonard
Schwartz, Mary
Vassallo, Mario

Artificial Intelligence (1985)

Allen, Kristen
Arora, Kulbir
Barback, Joseph
Baxter, William
Chen, Li-Wha
Clark, Michael
Hise, Denise
Hu, Hai Hsu
I, Chih-Li
Isaac, Reeba M
Kalnitz, Paul
Kramarczyk (Kramer), Chrisopher
Kuo, Chi-Kai
Li, Kuang Chieh
Lo, Mie-Ying
Lung, Hsi-Hao Howard
Min, Byoung Ho
Niyogi, Debashish
Palumbo, Paul
Pawlicki, Thadeus
Saks, Victor
Suchin, Jennifer
Taie, Shwu-Fan
Wang, Ching-Ying
Wang, Der-Yuk
Wiebe, Janyce
Yang, Jin-Tan David
Yi, Myungzoon

CS Other than AI (1985)

Chi, Henjin
Fu, Jing-sheng
Olin, David

Artificial Intelligence (1984)

Chang, Chung
Chen, Kwei-Jen
Choy, Chi Chung
Das, Mangobinda
Haefner, Michael
He, Hung Chyi
Hsu, An-Mei
Jou, Chen-Jye
Kellick, Diane
Konakanchi, Krishna
Kung, Peter F.
Leu, Fang Hsiung
Lin, Han-Hong
Liu, Ming
Liu, Peter (Sai-Ming)
Lo, Yu Li
Lung, Hsi-Hong
Milich, Gregory
Nemirov, Hinda
Phillips, Gretchen
Rapaport, William J.
Shlossman, Paul
Shen, Chien-Chih
Su, Suyuan C.
Yang, Lien Jang
Yao, Jo-Lan
Zayan, Ahme

CS Other than AI (1984)

Alsam, Javaid
He, Hung Chyi
Hung, Hing Kai
Izard, Thomas
Klee, Karl
Leung, Chun Wah
Oviedo, Enrique
Welte, Martha
Zachopoulos, George

3.16 ARTIFICIAL INTELLIGENCE FACULTY

Stuart C. Shapiro	Professor	Knowledge Representation Reasoning NL Processing
Sargur N. Srihari	Professor	Knowledge-Based Systems Computer Vision Pattern Recognition
William J. Rapaport	Associate Professor	Knowledge Representation Philosophical Foundations NL Processing
Shoshana Hardt	Assistant Professor	Expert Systems Qualitative Reasoning
David Sher	Assistant Professor	Computer Vision
Deborah K. W. Walters	Assistant Professor	Computer Vision
Richard Wildes	Assistant Professor	Computer Vision
Jonathan J. Hull	Research Assistant Professor	Computer Vision
Jeannette G. Neal	Research Assistant Professor Senior Scientist, Calspan	Intelligent Interfaces NL Understanding Expert Systems

3.17 ARTIFICIAL INTELLIGENCE ADDITIONS TO THE DEPARTMENT DURING THE PERIOD OF NAIC FUNDING

New Artificial Intelligence Faculty Appointed

Michael Leyton Assistant Professor	Computer Vision	8/86 - 8/87
David Sher Assistant Professor	Computer Vision	8/87 - present
Richard Wildes Assistant Professor	Computer Vision	9/88 - present
Jeannette G. Neal Research Assistant Professor Senior Scientist, Calspan	Intelligent Interfaces NL Understanding Expert Systems	2/88 - present
Jonathan J. Hull, Research Assistant Professor	Computer Vision	9/87 - present

New Artificial Intelligence Courses

CS 514 Vision
CS 666 Introduction to Image Analysis
CS 676 Knowledge Representation

3.18 ONGOING ARTIFICIAL INTELLIGENCE DISSERTATIONS

AI Students Past the Ph.D. Primary Area Examination
(Name in **bold** is currently supported by RADC)

Arora, Kulbir
Qualitative Reasoning about Physical Systems

Baxter, William
Multiresolution Edge Detection

Ehrlich, Karen
Automatic Acquisition of Natural Language

Kumar, Deepak
Planning in SNePS

Lively, Richard
Texture Segmentation of Images

Niyogi, Debashish
A Knowledge-Bases Approach to Analyzing Logical Document Structure

Pawlicki, Thaddeus F.
A Neural Network Approach to the Indexing Problems on Model-Based Computer Vision Systems

Srihari, Rohini
Integration of Information from Visual & Linquistic Sources

Wiebe, Janyce M.
A Computational Theory of Perspective in Narrative

Yuhan, Albert Hanyong
Dynamic Computation of Reference Frames in Spatial Information Processing

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.